EOxServer Documentation

Release 1.0.1

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

Jan 25, 2023

Contents

CHAPTER 1

Users' Guide

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.

1.1 EOxServer Basics

Table of Contents

1.1.1 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 install document for further information on installing the software.

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 for instructions how to subscribe.

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

[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

EOxServer Documentation

The EOxServer documentation consists of the

- *Users' Guide* (page 1) (which this document is part of)
- *Developers' Guide* (page 77) (where you can find implementation details)
- /rfc/index (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].

1.1.2 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 accompanyed 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 acquisistion footprint.

The data model is described in more detail in the *Coverages* (page 20) section.

1.1.3 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.

The following sectino gives an overview of the provided services, the full description can be found in the *Services* (page 30) chapter.

[10] https://github.com/EOxServer/eoxserver

Web Coverage Service

The OGC Web Coverage Service[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 .

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[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.

Web Processing Service

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

[11] http://www.opengeospatial.org/standards/wcs
[12] http://www.opengeospatial.org/standards/wms
[13] http://www.opengeospatial.org/standards/wps

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

1.2 Installation

This document is a guide to install EOxServer.

1.2.1 Installing from packages

EOxServer is packaged and distributed as a Python package. With that in prerequisite it is easy to define other Python dependencies. Unfortunately this is not the case for non-Python libraries, as they typically need to be installed via the operating systems package management system or some other means. Table: "*EOxServer Dependencies* (page 5)" below shows the minimal required software to run EOxServer.

Table 1: EOxServer Dependencies

Software	Required Version	Description
GDAL	>= 1.7.0	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.
MapServer	>= 7.0	Server software implementing various OGC Web Service interfaces including WCS and WMS. Includes a Python binding which is used by EOxServer.

When all non-python dependencies are installed, EOxServer can be installed using the `pip` (or sometimes `pip3`) utility.

```
# pip3 install -U eoxserver
```

In the default setting, this also fetches all Python package dependencies. The `-U` switch denotes that if EOxServer is already installed, it will be upgraded to the latest version.

If not otherwise packaged (like with Docker, see below), it is preferred to use a virtual environment

1.2.2 Using Docker images

If Docker is available, the easiest way to set up and use EOxServer is to use the pre-built and maintained docker images. The images can be obtained using the `docker pull` command like so:

```
# docker pull eoxa/eoxserver
Using default tag: latest
latest: Pulling from eoxa/eoxserver
93956c6f8d9e: Pull complete
46bddb84d1c5: Pull complete
15fa85048576: Pull complete
8aa40341c4fa: Pull complete
```

(continues on next page)

(continued from previous page)

```
7a299ef02497: Pull complete
09229f9ea579: Pull complete
3163f1230278: Pull complete
2f90ec943f31: Pull complete
12b403f83389: Pull complete
d6c5830b2cc6: Pull complete
658ea0984fee: Pull complete
7fbc330a1a79: Pull complete
Digest: sha256:7ec2310bf28074c34410fadb72c2c1b7dddbd6e381d97ce22ce0d738cd591619
Status: Downloaded newer image for eoxa/eoxserver:latest
docker.io/eoxa/eoxserver:latest
```

Note: This will fetch the image with the `latest` tag by default. Other tags using a different operating system or package versions may be available as well.

This image can now be started using the `docker run` command.

```
# docker run --rm -it -p 8000:8000 eoxa/eoxserver
```

As single docker containers are hard to control by themselves, other tools like Docker Compose can help to keep static and runtime configuration manageable.

Consider the following `docker-compose.yml` file:

```
version: "3.6"
services:
  database:
    image: mdillon/postgis:10
    volumes:
      - database-data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: "user"
      POSTGRES_PASSWORD: "pw"
      POSTGRES_DB: "dbms"
  eoxserver:
    image: eoxa/eoxserver
    environment:
      DB_USER: "user"
      DB_PW: "pw"
      DB_HOST: database
      DB_PORT: 5432
      DB_NAME: "dbms"
      XML_CATALOG_FILES: /opt/schemas/catalog.xml
    ports:
      - "8800:8000"

volumes:
  database-data:
```

This Docker Compose file can now be used to manage the database and EOxServer in a single step. The following command starts the services in the Compose file.

```
docker-compose up
```

The benefit of this approach is that with Docker Compose the services can resolve the other services by their names without having to deal with manual connection or hassling with IP addresses.

For production deployment, Docker Swarm is recommended instead.

1.3 Instance

EOxServer can only be used in an instantiated Django project. This instance incorporates the whole configuration necessary to run the web application. With this approach it is possible to deploy more than one web application per host.

1.3.1 Creation

An instance can be created in multiple ways. The easiest way is to run the `eoxserver-instance.py` script, that available through the EOxServer Python package, which has to be installed first. See the *Installation* (page 5) for more details.

Another option is to use the `django-admin` command to start a new Django project, that will later be enhanced to be a fully functioning EOxServer. See next section Configuration for what can be configured.

1.3.2 Configuration

The instance provides various different configuration files to configure the resulting web application. As each EOxServer instance is a Django Project at its core, it inherits all its configuration files.

These files are first and foremost the `settings.py` and `urls.py` files, but also the `wsgi.py` and `manage.py` to a lesser degree.

EOxServer uses the `settings.py` file to configure some of its internal functions. Please see the next section for the available sections and their effect.

Please see the Django Documentation for a coverage of the configuration capabilities.

1.3.3 Configurations in settings.py

These settings are used by Django directly, but are usually necessary do adapt:

PROJECT_DIR Absolute path to the instance directory.

DATABASES The database connection details. EOxServer requires a spatially enabled database backend. Both Spatialite and PostGIS are tested and known to work.

LOGGING what and how logs are prcessed and stored. EOxServer provides a very basic configuration that stores logfiles in the instace 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[14].

Please especially consider the setting of the TIME_ZONE[15] parameter and read the Notes provided in the `settings.py` file.

The following settings can be used to configure various parts of EOxServer.

EOXS_STORAGE_HANDLERS The enabled storage handlers as a list of paths to their respective implementing class.

Default:

[14] https://docs.djangoproject.com/en/2.2/topics/settings/
[15] https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-TIME_ZONE

```
[
    'eoxserver.backends.storages.ZIPStorageHandler',
    'eoxserver.backends.storages.TARStorageHandler',
    'eoxserver.backends.storages.DirectoryStorageHandler',
    'eoxserver.backends.storages.HTTPStorageHandler',
    'eoxserver.backends.storages.FTPStorageHandler',
    'eoxserver.backends.storages.SwiftStorageHandler',
]
```

EOXS_STORAGE_AUTH_HANDLERS The enabled storage authorization handlers as a list of paths to their respective implementing class.

Default:

```
[
    'eoxserver.backends.keystone.storage_auth.KeystoneStorageAuthHandler',
]
```

EOXS_MAP_RENDERER (="eoxserver.render.mapserver.map_renderer.MapserverMapRenderer") The map renderer to use for map rendering such as in WMS GetMap requests.

Default:

```
"eoxserver.render.mapserver.map_renderer.MapserverMapRenderer"
```

EOXS_MAPSERVER_LAYER_FACTORIES The list of layer factories for when the default MapServer map renderer is used.

Default:

```
[
    'eoxserver.render.mapserver.factories.CoverageLayerFactory',
    'eoxserver.render.mapserver.factories.OutlinedCoverageLayerFactory',
    'eoxserver.render.mapserver.factories.MosaicLayerFactory',
    'eoxserver.render.mapserver.factories.BrowseLayerFactory',
    'eoxserver.render.mapserver.factories.OutlinedBrowseLayerFactory',
    'eoxserver.render.mapserver.factories.MaskLayerFactory',
    'eoxserver.render.mapserver.factories.MaskedBrowseLayerFactory',
    'eoxserver.render.mapserver.factories.OutlinesLayerFactory',
]
```

EOXS_COVERAGE_METADATA_FORMAT_READERS The list of coverage metadata readers that will be employed to read metadata when a new coverage is registered.

Default:

```
[
    'eoxserver.resources.coverages.metadata.coverage_formats.gsc.GSCFormatReader',
    'eoxserver.resources.coverages.metadata.coverage_formats.dimap_general.
↪DimapGeneralFormatReader',
    'eoxserver.resources.coverages.metadata.coverage_formats.eoom.EOOMFormatReader
↪',
    'eoxserver.resources.coverages.metadata.coverage_formats.gdal_dataset.
↪GDALDatasetMetadataReader',
    'eoxserver.resources.coverages.metadata.coverage_formats.inspire.
↪InspireFormatReader',
    'eoxserver.resources.coverages.metadata.coverage_formats.native.NativeFormat',
    'eoxserver.resources.coverages.metadata.coverage_formats.native_config.
↪NativeConfigFormatReader',
```

(continues on next page)

(continued from previous page)

```
    'eoxserver.resources.coverages.metadata.coverage_formats.landsat8_l1.
↪Landsat8L1CoverageMetadataReader',
]
```

EOXS_COVERAGE_METADATA_GDAL_DATASET_FORMAT_READERS The list of coverage metadata readers that will be employed to read metadata when a new coverage is registered. These readers will use a GDAL dataset underneath.

Default:

```
[
    'eoxserver.resources.coverages.metadata.coverage_formats.gdal_dataset_envisat.
↪GDALDatasetEnvisatMetadataFormatReader',
]
```

EOXS_PRODUCT_METADATA_FORMAT_READERS The list of product metadata readers that will be employed to read metadata when a new product is registered.

Default:

```
[
    'eoxserver.resources.coverages.metadata.product_formats.sentinel1.
↪S1ProductFormatReader',
    'eoxserver.resources.coverages.metadata.product_formats.sentinel2.
↪S2ProductFormatReader',
    'eoxserver.resources.coverages.metadata.product_formats.landsat8_l1.
↪Landsat8L1ProductMetadataReader',
    'eoxserver.resources.coverages.metadata.coverage_formats.eoom.EOOMFormatReader
↪',
    'eoxserver.resources.coverages.metadata.product_formats.gsc.
↪GSCProductMetadataReader',
]
```

EOXS_MAPSERVER_CONNECTORS Default:

```
[
    'eoxserver.services.mapserver.connectors.simple_connector.SimpleConnector',
    'eoxserver.services.mapserver.connectors.multifile_connector.
↪MultiFileConnector',
    'eoxserver.services.mapserver.connectors.mosaic_connector.MosaicConnector',
]
```

EOXS_OPENSEARCH_FORMATS The list of OpenSearch result formats that shall be available for searching.

Default:

```
[
    'eoxserver.services.opensearch.formats.atom.AtomResultFormat',
    'eoxserver.services.opensearch.formats.rss.RSSResultFormat',
    'eoxserver.services.opensearch.formats.html.HTMLResultFormat',
    'eoxserver.services.opensearch.formats.kml.KMLResultFormat',
    'eoxserver.services.opensearch.formats.geojson.GeoJSONResultFormat',
]
```

EOXS_OPENSEARCH_EXTENSIONS The list of OpenSearch extension implementations.

Default:

```
[
    'eoxserver.services.opensearch.extensions.eo.EarthObservationExtension',
    'eoxserver.services.opensearch.extensions.geo.GeoExtension',
    'eoxserver.services.opensearch.extensions.time.TimeExtension',
    'eoxserver.services.opensearch.extensions.cql.CQLExtension',
]
```

EOXS_OPENSEARCH_SUMMARY_TEMPLATE (="opensearch/summary.html") The name of the template to use to generate the item summary.

Default:

```
"opensearch/summary.html"
```

EOXS_OPENSEARCH_RECORD_MODEL (="eoxserver.resources.coverages.models.EOObject") What record base model to use for OpenSearch searches. Can be set to `"eoxserver.resources.coverages.models.EOObject"`, `"eoxserver.resources.coverages.models.Coverage"`, or `"eoxserver.resources.coverages.models.Product"`. When using the generic EOObject the search can find both Products and Coverages, but the underlying query is significantly more complex, negatively impacting the performance.

Default:

```
"eoxserver.resources.coverages.models.EOObject"
```

EOXS_OWS_SERVICE_HANDLERS The enabled OWS service handlers. This configuration specifies what OWS services and versions are available for this instance.

Default:

```
[
    'eoxserver.services.ows.wcs.v10.handlers.GetCapabilitiesHandler',
    'eoxserver.services.ows.wcs.v10.handlers.DescribeCoverageHandler',
    'eoxserver.services.ows.wcs.v10.handlers.GetCoverageHandler',
    'eoxserver.services.ows.wcs.v11.handlers.GetCapabilitiesHandler',
    'eoxserver.services.ows.wcs.v11.handlers.DescribeCoverageHandler',
    'eoxserver.services.ows.wcs.v11.handlers.GetCoverageHandler',
    'eoxserver.services.ows.wcs.v20.handlers.GetCapabilitiesHandler',
    'eoxserver.services.ows.wcs.v20.handlers.DescribeCoverageHandler',
    'eoxserver.services.ows.wcs.v20.handlers.DescribeEOCoverageSetHandler',
    'eoxserver.services.ows.wcs.v20.handlers.GetCoverageHandler',
    'eoxserver.services.ows.wcs.v20.handlers.GetEOCoverageSetHandler',
    'eoxserver.services.ows.wms.v10.handlers.WMS10GetCapabilitiesHandler',
    'eoxserver.services.ows.wms.v10.handlers.WMS10GetMapHandler',
    'eoxserver.services.ows.wms.v11.handlers.WMS11GetCapabilitiesHandler',
    'eoxserver.services.ows.wms.v11.handlers.WMS11GetMapHandler',
    'eoxserver.services.ows.wms.v13.handlers.WMS13GetCapabilitiesHandler',
    'eoxserver.services.ows.wms.v13.handlers.WMS13GetMapHandler',
    'eoxserver.services.ows.wps.v10.getcapabilities.WPS10GetCapabilitiesHandler',
    'eoxserver.services.ows.wps.v10.describeprocess.WPS10DescribeProcessHandler',
    'eoxserver.services.ows.wps.v10.execute.WPS10ExecuteHandler',
    'eoxserver.services.ows.dseo.v10.handlers.GetCapabilitiesHandler',
    'eoxserver.services.ows.dseo.v10.handlers.GetProductHandler',
]
```

EOXS_OWS_EXCEPTION_HANDLERS The enabled OWS service exception handlers. This is similar to the service handlers, but defines how exceptions are encoded.

Default:

```
[
    'eoxserver.services.ows.wcs.v10.exceptionhandler.WCS10ExceptionHandler',
    'eoxserver.services.ows.wcs.v11.exceptionhandler.WCS11ExceptionHandler',
    'eoxserver.services.ows.wcs.v20.exceptionhandler.WCS20ExceptionHandler',
    'eoxserver.services.ows.wms.v13.exceptionhandler.WMS13ExceptionHandler',
]
```

EOXS_CAPABILITIES_RENDERERS The WCS capabilities renderers to use. Each one is tried with the given request parameters and the first fitting one is used.

Default:

```
[
    'eoxserver.services.native.wcs.capabilities_renderer.
↪NativeWCS20CapabilitiesRenderer',
    'eoxserver.services.mapserver.wcs.capabilities_renderer.
↪MapServerWCSCapabilitiesRenderer',
]
```

EOXS_COVERAGE_DESCRIPTION_RENDERERS The WCS coverage description renderers to use. For a DescribeCoverage request each implementation checked for compatibility and the first fitting one is used.

Default:

```
[
    'eoxserver.services.mapserver.wcs.coverage_description_renderer.
↪CoverageDescriptionMapServerRenderer',
    'eoxserver.services.native.wcs.coverage_description_renderer.
↪NativeWCS20CoverageDescriptionRenderer',
]
```

EOXS_COVERAGE_RENDERERS The WCS coverage renderers to use. For a GetCoverage request each implementation checked for compatibility and the first fitting one is used.

Default:

```
[
    'eoxserver.services.mapserver.wcs.coverage_renderer.
↪RectifiedCoverageMapServerRenderer',
    'eoxserver.services.gdal.wcs.referenceable_dataset_renderer.
↪GDALReferenceableDatasetRenderer',
]
```

EOXS_COVERAGE_ENCODING_EXTENSIONS Additional coverage encoding extensions to use.

Default:

```
[
    'eoxserver.services.ows.wcs.v20.encodings.geotiff.
↪WCS20GeoTIFFEncodingExtension'
]
```

EOXS_PROCESSES This setting defines what processes shall be available for WPS.

Default:

```
[
    'eoxserver.services.ows.wps.processes.get_time_data.GetTimeDataProcess'
]
```

EOXS_ASYNC_BACKENDS (=[]) The enabled WPS asynchronous backends. This setting is necessary to enable asynchronous WPS.

1.3.4 Configurations in `eoxserver.conf`

The `eoxserver.conf` uses the `.ini` file structure. This means the file is divided into sections like this: `[some.section]`. The following sections and their respective configuration keys are as follows:

[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 referencial coverages. This configuration option defaults to the systems standard[16].

[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.

[services.ows] This section entails various service metadata settings which are embedded in W*S GetCapabilities documents.

update_sequence=20131219T132000Z The service capabilities update sequence. This is used for clients to determine whether or not the service experienced updates since the last sequence.

name=EOxServer EO-WCS The service instance name.

title=Test configuration of MapServer used to demonstrate EOxServer The service instance title.

abstract=Test configuration of MapServer used to demonstrate EOxServer The service instance abstract/description.

onlineresource=http://eoxserver.org The service link.

keywords=<KEYWORDLIST> A comma separated list of keywords for this service.

fees=None Some additional information about service fees.

access_constraints=None Whether and how the service access is constrained.

provider_name=<CONTACTORGANIZATION> The service providing organizations name.

provider_site=<URL> The service providing organizations HTTP URL.

individual_name=<CONTACTPERSON> The main contact persons name.

position_name=<CONTACTPOSITION> The main contact persons position.

phone_voice=<CONTACTVOICETELEPHONE> The main contact persons voice phone number.

phone_facsimile=<CONTACTFACSIMILETELEPHONE> The main contact persons facsimile phone number.

[16] http://docs.python.org/library/tempfile.html#tempfile.mkstemp

electronic_mail_address=<CONTACTELECTRONICMAILADDRESS> The main contact persons email address.

delivery_point=<ADDRESS> The service providing organizations address.

city=<CITY> The service providing organizations city.

administrative_area=<STATEORPROVINCE> The service providing organizations province.

postal_code=<POSTCODE> The service providing organizations postal code.

country=<COUNTRY> The service providing organizations country.

hours_of_service=<HOURSOFSERVICE> The service providing organizations hours of service.

contact_instructions=<CONTACTINSTRUCTIONS> Additional contact instructions

role=Service provider The service providing organizations role.

[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 14)").

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 16)").

[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 14)").

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 16)").

[services.ows.wcs20]

paging_count_default=10 The maximum number of *wcs:coverageDescription* elements returned in a WCS 2.0 *EOCoverageSetDescription*. This also limits the *count parameter* (page 33). 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 14)").

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 14)").

maxsize=2048 The maximum size for each dimension in WCS GetCoverage responses. All sizes above will result in exception reports.

1.3.5 Setup

When your instance is configured, several steps need to be taken in order to set up the application. First off, the configured database needs to be migrated. This is achieved using the migrate[17] command. The following command performs the necessary migrations:

```
python manage.py migrate
```

Migration performs various steps depending on the necessity. For example it creates a database schema if it is not already present. If there already is a database schema, it is inspected to see whether it needs to be updated. If yes both the schema and the data already in the database will be updated.

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
```

1.4 Supported Raster File Formats and Their Configuration

Table of Contents

In this section, the EOxServer's handling of raster file formats and OWS service specific format configuration is described.

1.4.1 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[18] library. Thus, in principle, any raster file format supported by the GDAL[19] 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 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.

[17] https://docs.djangoproject.com/en/2.2/ref/django-admin/#django-admin-migrate
[18] http://www.gdal.org
[19] http://www.gdal.org/formats_list.html

1.4.2 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.

1.4.3 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 `serices.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 in invalid MIME-types (not listed among the supported formats) in `getCoverge()` requests will lead to errors (OWS Exceptions).

1.4.4 Web Coverage Service - Native Format Configuration

The *native format* (as defined by WCS 2.0.1 [OGC 09-110r4][20]) is the default raster file format returned by the `getCoverage()` operation in case of a missing explicit format specification. By default, EOxServer sets 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.

1.4.5 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 `serices.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).

1.4.6 References

[OGC 09-110r4] http://www.opengeospatial.org/standards/wcs

1.5 Supported CRSs and Their Configuration

Table of Contents

This section describes configuration of Coordinate Reference Systems for both WMS and WCS services.

[20] http://www.opengeospatial.org/standards/wcs

1.5.1 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[21].

1.5.2 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).

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 `serices.ows.wms`:

```
[services.ows.wms]
supported_crs= <EPSG-code>[,<EPSG-code>[,<EPSG-code> ... ]]
```

1.5.3 Web Coverage Service

EOxServer allows the specification of a list of CRCes 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 applicabe 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 `serices.ows.wcs`:

```
[services.ows.wcs]
supported_crs= <EPSG-code>[,<EPSG-code>[,<EPSG-code> ... ]]
```

1.6 Backends

The backends concepts provide a representation of data, metadata and other files that either reside on a local or remote storage.

The backends have a static representation in the database (i.e the data models) and a dynamic behavioral implementation: the handlers. The combintation of both allows the registration of storages, backend authorization and data items and the access at runtime.

[21] http://www.epsg-registry.org

1.6.1 Data model

The backends data model are represented by Django database models. The following classes provide both concrete and abstract model for the use of the other components of EOxServer.

Data Item

This abstract model is used to reference files, which are either local, or residing on a *Storage Model* (page 18). Each concrete implementation of this abstract class has at least a reference to a Storage, a location and an optional format specifier.

The `location` is always relative to the specified storage. When no storage is set, it is treated as a path to a local file.

Examples of concrete data items are the `ArrayDataItem` to store raster data for `Coverages` or the `MetaDataItem` to store arbitrary metadata for geospatial objects.

Storage

The Storage model allows to provide a simple abstraction of files on a remote storage or a local archive file. The type of the storage is denoted by its `storage_type` field. This value is used when accessing the storage via the `StorageHandler` class of the appropriate type.

Each storage has a `url` field that provides a basic "location" of the storage. The meaning of the field depends on the storage type. For an HTTP storage, for example, the URL would be the URL to the HTTP server and the root path for all data items to use, whereas for a ZIP file storage the URL is the path to the ZIP file.

Each storage can be given a name, which helps with management.

A Storage can be linked to a *Storage Auth* (page 18) model, which allows to specify authorization credentials.

Table 2: Default storage handlers

Storage type	Description
ZIP	ZIP file storage.
TAR	TAR file storage
directory	A local directory is treated as a storage file storage
HTTP	An HTTP server storage.
FTP	An FTP server storage.
swift	OpenStack swift object storage.

Storage Auth

The StorageAuth model stores authorization credentials. Similarly to the *Storage Model* (page 18) it is linked to a storage authorization handler class via its `storage_auth_type` attribute. The handler actually performs the authorization with the stored credentials. A typical example is the keystone authorization used for the OpenStack Swift object storage.

Table 3: Default storage auth handlers

Storage auth type	Description
keystone	Keystone client authorization. Requires the python-keystoneclient[22] and python-swiftclient[23] packages to be installed.

1.6.2 Command Line Management Interfaces

The following management commands provide facilities to manage the model instances related to the data backend.

storageauth This command provides two subcommands to `create` and `delete` Storage Auths.

create This sub-command allows to create a new Storage Auth. It requires the following arguments and supports the following options.

name the name of the Storage Auth to be created for internal reference

url the URL of the Storage Auth

--type, -t the type of the Storage Auth

--parameter, -p an additional parameter to set in the Storage Auth. Can be specified multiple times.

--check check if the access to the Storage Auth actually works. Raises an error if not.

The following example shows the creation of a keystone Storage Auth. The credentials are passed in as environment variables.

```
python manage.py storageauth create auth-cloud-ovh "${OS_AUTH_URL_SHORT}" \
    --type keystone \
    -p auth-version "${ST_AUTH_VERSION}" \
    -p identity-api-version="${ST_AUTH_VERSION}" \
    -p username "${OS_USERNAME}" \
    -p password "${OS_PASSWORD}" \
    -p tenant-name "${OS_TENANT_NAME}" \
    -p tenant-id "${OS_TENANT_ID}" \
    -p region-name "${OS_REGION_NAME}"
```

delete To delete a Storage Auth, the subcommand `delete` with the Storage Auth name must be passed. The following example deletes the previously created Storage Auth from above.

```
python manage.py storageauth delete auth-cloud-ovh
```

storage This command allows to manage storages. The subcommands `create`, `delete` allow to create new storages and delete no longer required ones.

create This sub-command creates a new storage. The following parameters and options can be passed.

name the storages name for internal reference

url the location reference. The actual meaning may change according to the storage type.

--type this is the string type of the storage. See the above table *Default Storage Handlers* (page 18) for the available ones.

--parent if the storage type supports parent storages, this parameter can be used to specify the parent storage. This allows to nest storages, e.g a ZIP archive on a HTTP server.

--storage-auth this parameter must be used for storage types that require additional authorization, such as OpenStack swift storages. Use the name of the Storage Auth as a value of this parameter.

The following example creates an OpenStack swift storage, linked to the Storage Auth created above.

[22] https://pypi.org/project/python-keystoneclient/
[23] https://pypi.org/project/python-swiftclient/

```
python manage.py storage create \
    MySwiftContainer container \
    --type swift \
    --storage-auth auth-cloud-ovh
```

delete This sub-command deletes a previously created storage.

name the name of the storage to delete

```
python manage.py storage delete MySwiftContainer
```

env This sub-command lists environment variables necessary to access the storage.

name the name of the storage to list the environment variables for

--path a path on the storage to list variables for

list A sub-command to list filenames on a storage

name the name of the storage to list files on

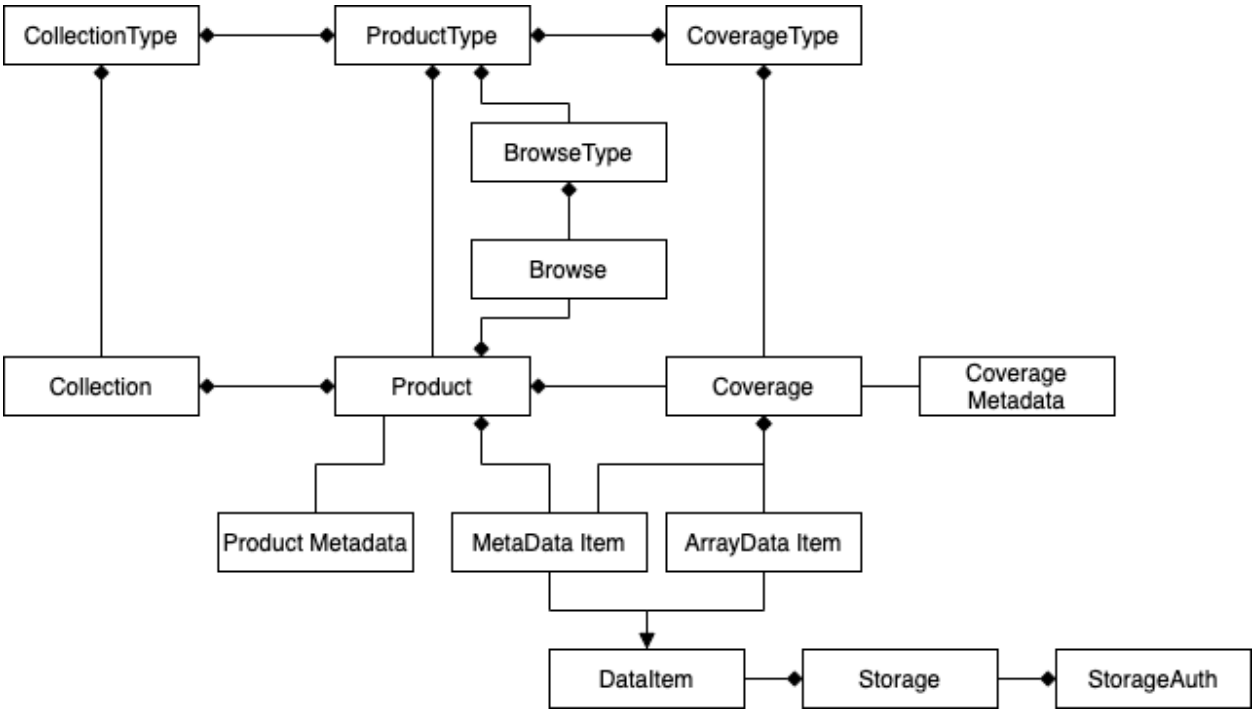
--pattern a file glob pattern to filter the returned filenames

--path a path on the storage to limit the file search

1.7 Coverages

This document describes the data model layout of the coverages, the internal structure of earth observation products, collections and data files. It also shows how these models can be interacted with via the command line interfaces.

1.7.1 Data model



The data model is loosely based on the OGC coverages data models, especially with the EO Application Profile for WCS 2.0.

Coverage Type

The coverage type describes the internal structure of coverages of a specific type. The coverage type is comprised of a list of field types that define the structure and metadata of a specific field of Data, called the Field Type.

The coverage type has a unique name to allow its identification.

Product Type

The product type model allows to define the structure of products by limiting the coverage types each coverage is allowed to have for products of this product type.

Additionally, each Product Type can be associated with a number of *Browse Type* (page 21) and *Mask Type* (page 21) that define the masks and browses that products of that type are allowed to have.

Browse Type

A browse type defines a typical visual representation of a product. For this purpose, it allows to define expression, scaling ranges and nodata values to generate that representation (browse).

A browse type can either define a single output band (grey), three output bands (RGB) or four output bands (RGBA).

Expressions must follow Python syntax rules but can only contain simple arithmetic expressions. All identifiers must be names of field types that are linked to coverage types in the list of allowed coverage types of the referenced product type.

Mask Type

These type models define what polygon masks can be linked to products of that product type and whether the masks define areas of validity or invalidity.

Collection Type

These type models allow to define the shape of collections by allowing to limit the product types and coverage types of product and coverages that can be added to collections of their respective collection types.

EOObject

This is the base model class for uniquely identifier geospatial objects. EOObject provides the fields `identifier` (mandatory and unique), the `footprint` (its geometry) and its temporal distribution: `begin_time` and `end_time`.

All objects inheriting from EOObject share a common pool of identifier. Thus, it is, for example, not possible for a collection to have the same identifier as a product or coverage.

Grid

A grid defines a regularly spaced grid comprised of up to four axes. Each axis can either be of spatial, temporal, evelation or other type. For each defined axis, the regular offset value must be specified.

Each grid is associated with a coordinate reference system.

A grid can be named, making it easier to manage.

A grid does *not* provide an actual location or area, this information can only be obtained with a Grid Fixture in conjunction with a grid.

Mosaic

This model is a collection of homogenenous coverages, all sharing the same coverage type and grid. This allows to access the mosaic as if it were a single coverage by combinig the data from all its comprising elements.

Coverage

A coverage is an n-dimensional raster data set comprised of several fields.

A coverage is linked to at least one ArrayDataItem, a reference to the actual raster data.

TODO: rel OGC coverage

Product

A product is a sort of collection of geospatially and temporally very close objects.

A product can combine multiple coverages which cover the same are but cannot be combined to a single coverage because of different resolutions.

In some cases, coverages are not necessary at all, and just provide data items for a binary download and browses for previewing.

Browse

A browse is always associated with a product and serves as a preview to the actual data. Browses are materialized files that are either pre-generated or can be generated from the coverage data.

Mask

Masks allow to specify regions in products for some kind of flag for example validity. Each mask is linked to a *Mask Type* (page 21).

Collection

Multiple coverages and products can be grouped in a collection. This relationship is many-to-many, so each product/coverage can be inserted into multiple collections.

When a collection is linked to a *Collection Type* (page 21) only Products and Coverages whose types are of the set of allowed coverage/product types can be inserted.

1.7.2 Command Line Interfaces

The following command line interfaces can be executed via the `manage.py` utility of the instance. All commands are related to one of the models above and use sub-commands for specific tasks.

coveragetype This command manages *Coverage Type* (page 21) models and allows to inspect the currently available ones.

create Creates a new Coverage Type with specifications from the parameters.

name the name of the Coverage type to create

--field-type add a new field type to the definition. Must be the five parameters: `identifier`, `description`, `definition`, `unit-of-measure`, and `wavelength`. Can be used multiple times to add more than one field.

TODO: example

import imports one or more Coverage Type definition from JSON files.

locations* a list of filenames to import definitions from

--in, -i read from `stdin` instead from a file

TODO: show definitition, example

delete deletes a Coverage Type

name the name of the Coverage Type to delete

--force, -f delete the Coverage Type, even if it is still in use. This cascades and deletes all Coverages of that type as well.

list lists the stored Coverage Types

--no-detail disable the printing of details of the coverage type.

producttype This command manages *Product Type* (page 21) models. It provides the following sub-commands:

create creates a new Product Type.

name the name of the Product Type to create

--coverage-type the Coverage Type name to add to this product type. Can be specified multiple times.

--mask-type the name of a to be created mask type.

--validity-mask-type the name of a to be created validity mask type.

--browse-type the name of a to be created Browse type. It is recommended to use `browsetype create` instead.

delete deletes a Product Type

name the name of the Product Type to delete

list lists all available Product Types

--no-detail disable the printing of details of the product type.

browsetype This command allows to create, delete and list *Browse Type* (page 21) models. Since Browse Types are always associated with a Product Type the first argument is always the name of a Product Type.

create creates a new Browse Type for a Product Type. Valid field names for the `--red`, `--green`, `--blue`, and `--alpha` parameters are the names from the field names of the linked Coverage Types of the associated Product Type.

product_type_name the Product Type to create the Browse Type for

[browse_type_name] the name of the Browse Type. Can be omitted, to define the default Browse Type.

--red, --grey, -r the field name or mathematical expression to use as the red output band (or grey, if used for a single band output).

--green, -g the field name or mathematical expression to use as the green output band.

--blue, -b the field name or mathematical expression to use as the blue output band.

--alpha, -a the field name or mathematical expression to use as the green output band.

--red-range, --grey-range the low and high border of values to apply a linear stretch for the red output band.

--green-range the low and high border of values to apply a linear stretch for the green output band.

--blue-range the low and high border of values to apply a linear stretch for the blue output band.

--alpha-range the low and high border of values to apply a linear stretch for the alpha output band.

--red-nodata, --alpha-nodata the nodata value for the red output band. This is applied after the stretch and will result in transparent pixels for this value.

--green-nodata the nodata value for the green output band. This is applied after the stretch and will result in transparent pixels for this value.

--blue-nodata the nodata value for the blue output band. This is applied after the stretch and will result in transparent pixels for this value.

--alpha-nodata the nodata value for the alpha output band. This is applied after the stretch and will result in transparent pixels for this value.

delete deletes a no longer needed Browse Type.

product_type_name the Product Type to delete the Browse Type from

[browse_type_name] the name of the Browse Type to delete

list lists all Browse Types for a given Product Type.

product_type_name the Product Type to list the Browse Types for

masktype This command allows to create, delete and list *Mask Type* (page 21) models. Since Mask Types are always associated with a Product Type the first argument is always the name of a Product Type. The sub-commands are in detail:

create creates a new Mask Type for a Product Type

product_type_name the Product Type to create the Mask Type for

mask_type_name the Mask Type name to create

--validity whether this mask denotes valid or invalid values. By default, it uses invalidity.

delete deletes a Mask Type.

product_type_name the Product Type to delete the Mask Type from

mask_type_name the Mask Type name to delete

list lists all Mask Types for a given Product Type.

product_type_name the Product Type to list the Mask Type of

collectiontype This command manages *Collection Type* (page 21) models using the following sub-commands:

create creates a new Collection Type.

name the name of the Collection Type

--coverage-type, -c the name of an existing Coverage Type, that shall be linked to this Collection Type. Only Coverages can be inserted into Collection when the Coverages Type is part of the Collections Type.

--product-type, -p the name of an existing Product Type, that shall be linked to this Collection Type. Only Products can be inserted into Collection when the Product Type is part of the Collections Type.

delete deletes a Collection Type.

name the name of the Collection Type to delete

--force, -f forces the deletion of all still existing Collections using this Collection Type.

list lists all available Collection Types.

--no-detail Disable the printing of details of the Collection types.

grid This command allows to create and delete named *Grid Model* (page 22) instances.

create this creates a Grid.

name the name of the Grid to create

coordinate_reference_system the definition of the coordinate reference system. Either an integer (the EPSG code), or the URL, WKT or XML definiton.

The following parameters can be used up to four times in order to define multiple axes.

--name, --axis-name, -n the name of the n-th axis to add to the Grid.

--type, --axis-type, -t the type of the n-th axis to add to the Grid.

--offset, --axis-offset, -o the fixed axis offset step of the n-th axis to add to the Grid.

delete deletes a Grid.

name the name of the Grid to delete.

coverage this command allows the registration and deregistration of *Coverage Model* (page 22) instances.

register this sub-command registers a Coverage.

--data, -d this specifies a location for raster data. Multiple values can be used to denote that the data resides on a storage. If used in that way the first value can also be the name of a named storage. This parameter can be used multiple times, when the raster data is split into multiple files.

--meta-data, -m similarly to the `--data` parameter, this parameter denotes a reference to meta-data. The same rules as for the `--data` parameter also apply here.

--type, --coverage-type, -t specify the *Coverage Type* (page 21) for this Coverage. By default no Coverage Type is used.

--grid, -g specify the named *Grid Model* (page 22) to use. By default an anonymous Grid is used with the CRS of the raster data files.

--size, -s specifies the size of the Coverage. This overrides the size extracted from the metadata/data. Must specify the size for each axis of the Grid.

--origin, -o overrides the origin of the Coverage. Must provide a value for each axis of the Grid.

--footprint, -f overrides the geographical footprint of the Coverage. Must be a valid WKT geometry.

--footprint-from-extent The footprint polygon shall be calculated from the Coverages extent.

--identifier, -i override the Coverages identifier.

--identifier-template allows the construction of the final identifier from a template. Substitution values are passed in from the extracted metadata. e.g: `{identifer}__B01`.

--begin-time, -b override the begin timestamp of the Coverage. Must be a valid ISO 8601 datetime string.

--end-time, -e override the end timestamp of the Coverage. Must be a valid ISO 8601 datetime string.

--product, --product-identifier, -p specify the Product identifier this Coverage shall be associated with. The Product must already be registered.

--collection, --collection-identifier, -c specify the Collection identifier this Coverage shall be inserted into. The Collection must already exist.

--replace, -r replace an already existing Coverage with the same identifier.

--use-subdatasets, --subdatasets specify to interpret colons in the filename as subdataset specifiers.

--print-identifier this switch prints the final identifier (after metadata extraction and potential templating) to stdout upon successful registration.

deregister this sub-command de-registers the Coverage with the provided identifier.

identifier the Coverages identifier

--not-refresh-collections this command will update all Collections metadata (footprint, begin-/end time) unless this switch is set.

--all, -a When this flag is set, all the Coverages are selected to be derigesterd.

product this command manages *Product Model* (page 22) instances.

register this sub-command registers products.

--identifier, -i override the Product identifier.

--identifier-template allows the construction of the final identifier from a template. Substitution values are passed in from the extracted metadata. e.g: `{identifer}__B01`.

--footprint overrides the geographical footprint of the Product. Must be a valid WKT geometry.

--begin-time override the begin timestamp of the Product. Must be a valid ISO 8601 datetime string.

--end-time override the end timestamp of the Product. Must be a valid ISO 8601 datetime string.

--set, -s sets a specific metadata value for that product. This parameter always uses two values: the name of the parameter key and its value. TODO: possible metadata keys to set

--metadata-file adds a metadata file to the product. As with file links for Coverages, the product file can be located on a storage. For these cases, multiple values can be used to specify the chain of locations.

--type, --product-type, -t specify the *Product Type* (page 21) for this Product. By default no Product Type is used.

--mask, -m specify a mask file to be added to this product. Must be two values: the masks name and its file location.

--mask-geomety, -g specify a mask using its geometry directly. Must be two values: the masks name and its WKT geometry representation.

--no-extended-metadata when this flag is set, only the basic metadata (identifier, footprint, begin- and end-time) is stored.

--no-masks when this flag is set, no masks will be discovered.

--no-browses when this flag is set, no browses will be discovered.

--no-metadata when this flag is set, no metadata files will be discovered.

--package specify the main data package for this Product.

--collection, --collection-identifier, -c specify the Collection identifier this Product shall be inserted into. The Collection must already exist.

--replace replace an already existing Product with the same identifier.

--print-identifier this switch prints the final identifier (after metadata extraction and potential templating) to stdout upon successful registration.

deregister deregisters a Product.

identifier the identifier of the Product to deregister.

--all, -a When this flag is set, all the Coverages are selected to be derigesterd.

discover print the contents of the main package file of a Product.

identifier the identifier of the Product to discover.

[pattern] a filename glob pattern to filter the resulting filenames

browse this command allows to manage *Browse Model* (page 22) instances of a *Product Model* (page 22).

register this sub-command registers a Browse to a Product.

identifier the Product identifier to register the Browse for.

location the storage location of the Browse.

--type the Browse Type name of that Browse.

generate TODO

deregister TODO

mask this command allows to manage *Mask Model* (page 22) instances of a *Product Model* (page 22).

register registers a Mask for a Product.

identifier the Product identifier to register the Mask for.

--type the Mask Type name of that Mask.

--location the storage location of the Mask.

--geometry the inline WKT geometry for the mask.

deregister_parser deregisters a Mask from a Product

identifier the Product identifier to deregister the Mask from.

collection this command manages *Collection Model* (page 22) instances. As usual, it uses sub-commands to allow fine control over the specific aspects and tasks of a Collection.

create creates a new Collection.

identifier the identifier for the new Collection.

--type, -t specify a Collection Type for this new Collection.

--grid, -g specify a Grid for this Collection.

--set, -s set or override Collection metadata. TODO: what keys?

delete this sub-command deletes a Collection.

identifier the identifier of the Collection to delete

--all, -a When this flag is set, all the collections are selected to be derigesterd.

insert with this sub-command one or more *Coverage Model* (page 22) instances or *Product Model* (page 22) instances can be inserted into the collection. This command checks whether the to be inserted objects are of the allowed types when a Collection Type is set for this Collection.

identifier the identifier of the Collection to insert objects into.

object_identifiers+ the list of object identifiers (either Products or Coverages) to insert into the Collection.

exclude this command allows to remove one or more objects from a collection.

identifier the identifier of the Collection to exclude objects from.

object_identifiers+ the list of object identifiers (either Products or Coverages) to exclude from the Collection.

purge this command purges all Coverages and Products from this Collection, leaving it effectively empty.

TODO: not yet implemented

summary collects metadata from all entailed Products and Coverages to generate a summary that is stored in the Collection. This allows a quick overview of the metadata ranges and specific values of all objects in the collection.

identifier the Collection identifier to generate the summary for

–products/–no-products whether or not to generate a Product metadata summary.

–coverages/–no-coverages whether or not to generate a Coverage metadata summary.

mosaic this command manages *Mosaic Model* (page 22) instances with a variety of sub-commands.

create creates a new Mosaic.

identifier the identifier of the Mosaic to create.

--type, -t the Coverage Type name for the Mosaic to create.

--grid, -g the Grid to use for the Mosaic.

delete deletes a Mosaic.

identifier the identifier of the Mosaic to delete.

insert insert one or more Coverages into the Mosaic.

identifier the identifier of the Mosaic to insert Coverages into.

coverage_identifiers+ the Coverage identifiers to insert into the Mosaic.

exclude exclude one or more Coverages from the Mosaic.

identifier the identifier of the Mosaic to exclude Coverages from.

coverage_identifiers+ the Coverage identifiers to exclude from the Mosaic.

refresh refresh the summary metadata of the Mosaic.

identifier the identifier of the Mosaic to generate the metadata.

purge TODO not implemented

id this command allows to introspect the contents of the instances database.

check this subcommand allows to check whether or not an object is registered. The return value of this command indicates whether such an object exists.

identifiers+ the identifier(s) to check for existence.

--type, -t limit the check to the given object type (i.e: `Coverage`, `Product`, `Collection`, or `Mosaic`). By default the search is for any `EOObject`.

list this command lists the contents of the database and prints the objects on on the terminal. Filters can be applied to limit the search.

identifiers* limit the output to the given identifiers.

--type, -t limit the listing to the given object type (i.e: `Coverage`, `Product`, `Collection`, or `Mosaic`). By default the search is for any `EOObject`.

--recursive, -r do a recursive lookup into the given collections.

--suppress-type, -s when printing an object, suppress the type and only print the identifier

--collection, -c limit the search to this collection only. Can be passed multiple times to search across multiple collections.

mapcache this command allows to generate an index database to be used for mapcache time dimensions.

sync this sub-command synchronizes a mapcache index database. The output will be written to the `<collection-name>.sqlite` files for each available collection in the current working directory.

The schema of the database will be the following:

```
CREATE TABLE "time" (
    "start_time" timestamp with time zone NOT NULL,
    "end_time" timestamp with time zone NOT NULL,
    "minx" double precision NOT NULL,
    "miny" double precision NOT NULL,
    "maxx" double precision NOT NULL,
    "maxy" double precision NOT NULL
)
```

--force, -f force the re-generation of the index files.

--unique-times, -u force unique time entries. This combines the extent of all objects with overlapping time spans.

--no-index this flag prohibits the creation of an internal database index.

stac This command allows to register Products and their related data from **'STAC Items'_**.

register this sub-command registers a STAC Item as a Product and its raster data as Coverages.

--in, -i Read the STAC Item from stdin instead from a file.

--type TYPE_NAME, --product-type TYPE_NAME, -t TYPE_NAME The name of the product type to associate the product with. Optional.

--replace, -r Optional. If the product with the given identifier already exists, replace it. Without this flag, this would result in an error.

types this sub-command extracts all the relevant information to generate Product Types, Coverage Types and their related types to allow a subsequent registration.

--in, -i read the STAC Item from stdin instead from a file.

--type TYPE_NAME, --product-type TYPE_NAME, -t TYPE_NAME the name of the new product type. Optional.

1.8 Services

1.8.1 Web Coverage Service (WCS)

A Web Coverage Service (WCS) offers multi-dimensional coverage data for access over the Internet.

The standard can be obtained from the Open Geospatial Consortiums homepage[24].

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

GetCapabilities

Table: "*WCS GetCapabilities Request Parameters* (page 31)" below lists all parameters that are available with Capabilities requests.

[24] https://www.ogc.org/standards/wcs

Table 4: 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
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[,...]]	• 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: "*WCS DescribeCoverage Request Parameters* (page 31)" below lists all parameters that are available with DescribeCoverage requests.

Table 5: 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): • valid coverageID of a Dataset • valid coverageID of a StichedMosaic		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 33)" below lists all parameters that are available with DescribeEOCoverageSet requests.

Table 6: 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: • using the coverageId of a Datatset • using the eoId of a DatatsetSeries • 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: • containment = overlaps (default) • containment = contains Any combination thereof (but each value only once per request)	• 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 *subset* parameter	• overlaps (default) • contains	O
section	see GetCapabilities	• 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 35)" below lists all parameters that are available with GetCoverage requests.

Table 7: 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): • valid coverageID of a Dataset • valid coverageID of a StichedMosaic		M
format	Requested format of coverage to be returned. By default the coverage is returned in its original format.	image/tiff	O
mediatype	Coverage delivered directly as image file or enclosed in GML structure • not present or • multipart/mixed	multipart/mixed	O
subset	Trimming of coverage dimension (no slicing allowed!) • the label of a coverage axis – The meaning of the subset can be altered by the subsettingCrs parameter.	• 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 • not present or • CRS	http://www.opengis.net/def/crs/EPSG/0/3035	O
rangesubset	Subsetting in the range domain (e.g. Band-Subsetting).	• Blue,Green,Red • Band1:Band3,Band5,Band7:Band9	O
scaleFactor	Scale the output by this factor. The 'scaleFactor' parameter requires MapServer v7.0.	• 0.5 • 1.25	O
• scaleAxes • scaleSize	Mutually exclusive per axis, either:	• scaleAxes=x(1.5),y(0.5)	O

1.8.2 Web Map Service (WMS)

> The OpenGIS® Web Map Service Interface Standard (WMS) provides a simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases. A WMS request defines the geographic layer(s) and area of interest to be processed. The response to the request is one or more geo-registered map images (returned as JPEG, PNG, etc) that can be displayed in a browser application. The interface also supports the ability to specify whether the returned images should be transparent so that layers from multiple servers can be combined or not.

The standard can be obtained from the Open Geospatial Consortiums homepage[26].

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

GetCapabilities

Table: "*WMS GetCapabilities Request Parameters* (page 36)" below lists all parameters that are available with Capabilities requests.

Table 8: WMS GetCapabilities Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
service	Requested service	WMS	M
request	Type of request	GetCapabilities	M
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[,...]]	1.3.0, 1.1.0, 1.0.0	O
updateSequence	Date of last issued GetCapabilities request; to receive new document only if it has changed since	"2013-05-08"	O

GetMap

Table: "*WMS GetMap Request Parameters* (page 37)" below lists all parameters that are available with GetMap requests.

[2] Interpolation: (Note: Resampling options other than NEAREST can dramatically slow down raster processing). The default (and fastest) is NEAREST. Replaces the target pixel with its NEAREST Neighbor. AVERAGE will compute the average pixel value of all pixels in 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 specificaiton of this encoding extension can be found here[25]

[25] https://portal.opengeospatial.org/files/?artifact_id=54813

[26] https://www.ogc.org/standards/wms

[1] For WMS service version 1.3 the `crs` parameter must be used, for services versions below 1.3 the parameter name is `srs`.

Table 9: WMS GetMap Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
service	Requested service	WMS	M
request	Type of request	GetMap	M
version	Version number	1.3.0, 1.1.0, 1.0.0	M
layers	The layers to render. Must be a comma-separated list of layer names. Exposed layers are listed in the Capabilities document and depend on the contents of the instance. For each object in the database a base layer with the objects identifier as a name is added. Additionally a number of layers are added with the objects identifier plus a postfix as show in the list below: • all: – `<no-postfix>`: the default rendering of the object – `outlines`: the objects footprint as a rendered geometry – `outlined`: the default rendering of the object overlayed with the objects rendered footprint. • `Collection/Product`: – `<mask-name>`: the rendered mask geometries for all products in that collection or that single product. – `masked_<mask-name>`: the default rendering of each product, each individually masked with		M

Layer Mapping

Various objects in EOxServer generate exposed layers to be requested by clients via WMS.

Table 10: WMS Layer Mapping

Base Object	Suffix	Description	Style	Advertised [2]
Coverage	–	Renders the coverage as a map. This is the most basic for of rendering and `dim_bands` and `dim_range` will likey need to be used to achieve representative result.	When the coverage only has a single field, or only one is selected via `dim_bands`, then the name of a color scale can be passed to colorize the otherwise greyscale image.	no
Mosaic	–	This behaves exactly like with Coverages but applies the rendering to all contained Coverages.	Same as above.	yes
Product	–	Renders the Products default Browse or using the defaults Browse Type to dynamically render a browse.		no
Coverage/Product	`outlines`	Renders the footprint of the Coverage/Product as a colorized geometry.	Defines the color of the rendered geometry.	no
Mosaic/Collection	`outlines`	Renders the footprint of all contained Coverages or Products as a colorized geometry.	Defines the color of the rendered geometry.	yes
Coverage/Product	`outlined`	Renders the Coverage/Product in its default way (as with no prefix) but overlays it with the footprint geometry (as with `outlines` suffix)	Defines the color of the rendered geometry.	no
Mosaic/Collection	`outlined`	Renders the Mosaic/Collection in its default way (as with no prefix) but each included Coverage/Product rendering is overlayed with the footprint geometry (as with the `outlines` suffix).	Defines the color of the rendered geometry.	yes
Product	`<Browse Type Name>`	Renders the Products Browse of that Browse Type if available or uses the Browse Type to dynamically render a Browse.		no
Product	`<Mask Type Name>`	Renders the Mask of the Product of that Mask Type as a rasterized vector layer.	Defines the color of the geometry.	no
Product	`masked_<Mask Type Name>`	Uses the default rendering of the product and apply the Mask of the specified Mask Type.		no
Collection	–	Renders all Products in the Collection with their default Browse (or dynamically using the default Browse Type).		
Collection	`<Browse Type Name>`	Renders all contained Products using the Browse of that Browse Type or dynamically generated Browse of that Browse Type.		
Collection	`<Mask Type Name>`	Renders all Masks of the contained Products as colorized geometries.		
Collection	`masked_<Mask Type Name>`	Renders all contained Browses using their default Browse or a dynamically generated Browse of the default Browse Type and individually apply the Mask of that Mask Type.		

1.8.3 Web Processing Service (WPS)

> The OpenGIS® Web Processing Service (WPS) Interface Standard provides rules for standardizing how inputs and outputs (requests and responses) for geospatial processing services, such as polygon overlay. The standard also defines how a client can request the execution of a process, and how the output from the process is handled. It defines an interface that facilitates the publishing of geospatial processes and clients' discovery of and binding to those processes. The data required by the WPS can be delivered across a network or they can be available at the server.

The standard can be obtained from the Open Geospatial Consortiums homepage[27].

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

GetCapabilities

Table: "*WPS GetCapabilities Request Parameters* (page 40)" below lists all parameters that are available with Capabilities requests.

Table 11: WPS GetCapabilities Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
service	Requested service	WPS	M
request	Type of request	GetCapabilities	M
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[,...]]	1.0.0	O
updateSequence	Date of last issued GetCapabilities request; to receive new document only if it has changed since	"2013-05-08"	O

DescribeProcess

Table: "*WPS DescribeProcess Request Parameters* (page 41)" below lists all parameters that are available with GetCoverage requests.

[27] https://www.ogc.org/standards/wps

[1] For WMS service version 1.3 the `crs` parameter must be used, for services versions below 1.3 the parameter name is `srs`.

Table 12: WPS DescribeProcess Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
service	Requested service	WPS	M
request	Type of request	DescribeProcess	M
version	Version number	1.0.0	M
identifier	The process identifier to get a detailed description for. It is possible to get multiple descriptions by passing a comma separated list of process identifiers. The process identifiers can be obtained from the GetCapabilities document.		M

Execute

Table: "*WPS Execute Request Parameters* (page 41)" below lists all parameters that are available with GetCoverage requests.

Table 13: WPS Execute Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
service	Requested service	WPS	M
request	Type of request	Execute	M
version	Version number	1.0.0	M
identifier	The process to execute.		M
DataInputs	A key-value mapping of data inputs. For each input, the unit of measure (UOM)	input1=abc@uom:a	M
ResponseDocument	This parameter selects the outputs of interest, their format and unit of measure (UOM).	output1=abc@uom:a	O
RawDataOutput	Selects a single output that shall be returned as a raw data item. Mutually exclusive with `ResponseDocument`.	input1=abc@uom:a	O
status	Boolean value whether to include a data lineag in the response document.		O
lineage	Boolean value whether to include a data lineag in the response document.		M
storeExecuteResponse	Boolean value whether to store the result on the server.		O

1.8.4 Download Service for Earth Observation Products (DSEO)

The Download Service for Earth Observation Products is an OGC best practice document to allow the download of earth observation products. The document can be obtained from the Open Geospatial Consortiums homepage[28].

[28] https://portal.opengeospatial.org/files/?artifact_id=55210

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

GetCapabilities

Table: "*DSEO GetCapabilities Request Parameters* (page 42)" below lists all parameters that are available with Capabilities requests.

Table 14: DSEO GetCapabilities Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
service	Requested service	DSEO	M
request	Type of request	GetCapabilities	M
acceptVersions	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[,...]]	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	• All • ServiceIdentification • ServiceProvider • OperationsMetadata	O
updateSequence	Date of last issued GetCapabilities request; to receive new document only if it has changed since	"2013-05-08"	O

GetProduct

Table: "*DSEO GetProduct Request Parameters* (page 42)" below lists all parameters that are available with GetProduct requests.

Table 15: DSEO GetProduct Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
service	Requested service	DSEO	M
request	Type of request	GetProduct	M
version	Version number	1.0.0	M
productUri	Valid identifier of a registered Product		M

This request downloads the product as a packaged file. If available, the Products referenced `package` is forwarded. Otherwise, all files of the Product and its referenced Coverages are packaged into a ZIP file which is then sent to the client.

1.8.5 OpenSearch

Table of Contents

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, Time[29] and Geo[30] extensions are implemented to limit the spatio-temporal scope of the search. Additionally, EO[31] extension is implemented to support most of the required and recommended best practices of the CEOS OpenSearch Best Practice Document[32].

Setup

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

```
from django.urls import include, re_path

urlpatterns = [
    ...
```

(continues on next page)

[29] http://www.opensearch.org/Specifications/OpenSearch/Extensions/Time/1.0/Draft_1
[30] http://www.opensearch.org/Specifications/OpenSearch/Extensions/Geo/1.0/Draft_2
[31] https://docs.opengeospatial.org/is/13-026r8/13-026r8.html
[32] https://earthdata.nasa.gov/files/CEOS_OpenSearch_Best_Practice_Doc-v.1.0.1_Jun2015.pdf

(continued from previous page)

```
    re_path(r'^opensearch/', include('eoxserver.services.opensearch.urls')),
    ...
)
```

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.

The `EOXS_OPENSEARCH_FORMATS`, `EOXS_OPENSEARCH_EXTENSIONS`, `EOXS_OPENSEARCH_SUMMARY_TEMPLATE`, and `EOXS_OPENSEARCH_RECORD_MODEL` settings in the `settings.py` alter the behavior of the service. The details can be found in the instance configuration section.

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 end 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 85), 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?}&amp;count={count?}&amp;startIndex={startIndex?}&
↪amp;bbox={geo:box?}&amp;geom={geo:geometry?}&amp;lon={geo:lon?}&amp;lat={geo:lat?}&
↪amp;r={geo:radius?}&amp;georel={geo:relation?}&amp;uid={geo:uid?}&amp;start=
↪{time:start?}&amp;end={time:end?}&amp;timerel={time:relation?}"/>
  <Url type="application/vnd.geo+json" rel="collection" template="http://localhost/
↪opensearch/json/?q={searchTerms?}&amp;count={count?}&amp;startIndex={startIndex?}&
↪amp;bbox={geo:box?}&amp;geom={geo:geometry?}&amp;lon={geo:lon?}&amp;lat={geo:lat?}&
↪amp;r={geo:radius?}&amp;georel={geo:relation?}&amp;uid={geo:uid?}&amp;start=
↪{time:start?}&amp;end={time:end?}&amp;timerel={time:relation?}"/>
  <Url type="application/vnd.google-earth.kml+xml" rel="collection" template="http://
↪localhost/opensearch/kml/?q={searchTerms?}&amp;count={count?}&amp;startIndex=
↪{startIndex?}&amp;bbox={geo:box?}&amp;geom={geo:geometry?}&amp;lon={geo:lon?}&amp;
↪lat={geo:lat?}&amp;r={geo:radius?}&amp;georel={geo:relation?}&amp;uid={geo:uid?}&
↪amp;start={time:start?}&amp;end={time:end?}&amp;timerel={time:relation?}"/>
  <Url type="application/rss+xml" rel="collection" template="http://localhost/
↪opensearch/rss/?q={searchTerms?}&amp;count={count?}&amp;startIndex={startIndex?}&
↪amp;bbox={geo:box?}&amp;geom={geo:geometry?}&amp;lon={geo:lon?}&amp;lat={geo:lat?}&
↪amp;r={geo:radius?}&amp;georel={geo:relation?}&amp;uid={geo:uid?}&amp;start=
↪{time:start?}&amp;end={time:end?}&amp;timerel={time:relation?}"/>
```

(continues on next page)

(continued from previous page)

```
  <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 49). 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&amp;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?}&amp;count=
↪{count?}&amp;startIndex={startIndex?}&amp;bbox={geo:box?}&amp;geom={geo:geometry?}&
↪amp;lon={geo:lon?}&amp;lat={geo:lat?}&amp;r={geo:radius?}&amp;georel={geo:relation?}
↪&amp;uid={geo:uid?}&amp;start={time:start?}&amp;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?}&amp;count=
↪{count?}&amp;startIndex={startIndex?}&amp;bbox={geo:box?}&amp;geom={geo:geometry?}&
↪amp;lon={geo:lon?}&amp;lat={geo:lat?}&amp;r={geo:radius?}&amp;georel={geo:relation?}
↪&amp;uid={geo:uid?}&amp;start={time:start?}&amp;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?}&amp;startIndex={startIndex?}&amp;bbox={geo:box?}&amp;geom=
↪{geo:geometry?}&amp;lon={geo:lon?}&amp;lat={geo:lat?}&amp;r={geo:radius?}&amp;
↪georel={geo:relation?}&amp;uid={geo:uid?}&amp;start={time:start?}&amp;end={time:end?
↪}&amp;timerel={time:relation?}"/>
  <Url type="application/rss+xml" rel="results" template="http://localhost/opensearch/
↪collections/MER_FRS_1P_reduced_RGB/rss/?q={searchTerms?}&amp;count={count?}&amp;
↪startIndex={startIndex?}&amp;bbox={geo:box?}&amp;geom={geo:geometry?}&amp;lon=
↪{geo:lon?}&amp;lat={geo:lat?}&amp;r={geo:radius?}&amp;georel={geo:relation?}&amp;
↪uid={geo:uid?}&amp;start={time:start?}&amp;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.
```

(continues on next page)

(continued from previous page)

```
]
}
```

EO Extension

Since version 0.4 EOxServer prvides implementation of the OpenSearch EO[33] extension. This extension supports most of the required and recommended best practices of the CEOS OpenSearch Best Practice Document[34].

The EO extension allows the following EO parameters to be added to the Opensearch request:

Table 16: OpenSearch Search Request EO Parameters

Parameter (Replacement Tag)	Description	Example
productType (eop:productType)	A string that identifies the product type.	productType=GES_DISC_AIRH3STD_V005
doi (eo:doi)	A Digital Object Identifier "string" identifying the product in the DOI[35] system.	doi=doi:10.7666/d.y351065
platform (eo:shortName)	The platform / satellite short name.	platform=Sentinel-1
platformSerialIdentifier (eo:serialIdentifier)	The Platform / satellite serial identifier.	
instrument (eop:shortName)	The name of the sensor / instrument.	instrument=ASAR
sensorType (eo:sensorType)	The sensor type.	sensorType=ATMOSPHERIC
compositeType (eo:compositeType)	The type of composite product expressed as time period that the composite product covers.	compositeType=P10D (P10D) is for 10 days coverage period
processingLevel (eo:processingLevel)	The processing level applied to the product.	
orbitType (eo:orbitType)	The platform / satellite orbit type.	orbitType=LEO (low earth orbit)
spectralRange (eo:spectralRange)	The sensor spectral range.	spectralRange= INFRARED
wavelengths (eo:discreteWavelengths)	A number, set or interval requesting the sensor wavelengths in nanometers.	
hasSecurityConstraints	A text informs if the resource has any security constraints. Possible values: TRUE, FALSE	hasSecurityConstraints=FALSE
dissemination	The dissemination method.	dissemination=EUMETCast
recordSchema	Metadata model in which additional metadata should be provided inline.	
parentIdentifier (eo:parentIdentifier)	The parent of the entry in a hierarchy of resources.	
productionStatus (eo:status)	The status of the entry.	productionStatus=ARCHIVED

Continued on next page

[33] https://docs.opengeospatial.org/is/13-026r8/13-026r8.html
[34] https://earthdata.nasa.gov/files/CEOS_OpenSearch_Best_Practice_Doc-v.1.0.1_Jun2015.pdf

Table 16 – continued from previous page

Parameter (Replacement Tag)	Description	Example
acquisitionType (eo:acquisitionType)	Used to distinguish at a high level the appropriateness of the acquisition for "general" use, whether the product is a nominal acquisition, special calibration product or other. Values: NOMINAL, CALIBRATION, OTHER.	acquisitionType=CALIBRATION
orbitNumber (eo:orbitNumber)	A number, set or interval requesting the acquisition orbit.	
orbitDirection (eo:orbitDirection)	the acquisition orbit direction.	orbitDirection=ASCENDING
track (eo:wrsLongitudeGrid)	the orbit track.	
frame (eo:wrsLatitudeGrid)	the orbit frame.	
swathIdentifier (eo:swathIdentifier)	Swath identifier. Value list can be retrieved with codeSpace.	swathIdentifier=I3 (Envisat ASAR has 7 distinct swaths (I1,I2...I7) that correspond to precise incidence angles for the sensor)
cloudCover (eo:cloudCoverPercentage or eo:cloudCoverPercentage)	The cloud coverage percantage.	cloudCover=65
snowCover (eo:snowCoverPercentage or eo:snowCoverPercentage)	The cloud coverage percantage.	cloudCover=65
lowestLocation (eo:lowestLocation)	The bottom height of datalayer (in meters).	
highestLocation (eo:highestLocation)	The top height of datalayer (in meters).	
productVersion (eo:version)	The version of the Product.	
productQualityStatus (eo:productQualityDegradation)	An optional field that must be provided if the product passed a quality check. Possible values: NOMINAL and DEGRADED.	productQualityStatus=DEGRADED
productQualityDegradationTag (eo:productQualityDegradationTag)	The degradations affecting the product.Possible values are mission specific and can be freely defined.	productQualityDegradationTag=RADIOMETRY
processorName (eo:processorName)	The processor software name.	
processingCenter (eo:processingCenter)	The processing center.	processingCenter=PDHS-E
creationDate (eo:creationDate)	The date when the metadata item was ingested for the first time (i.e. inserted) in the catalogue.	
modificationDate (eo:modificationDate)	The date when the metadata item was last modified (i.e. updated) in the catalogue.	
processingDate (eo:processingDate)	A date interval requesting entries processed within a given time interval.	
sensorMode (eo:operationalMode)	The sensor mode.	
archivingCenter (eo:archivingCenter)	The the archiving center.	

Continued on next page

Table 16 – continued from previous page

Parameter (Replacement Tag)	Description	Example
processingMode (eo:ProcessingMode)	Processing mode. Often referred to as Real Time, Near Real Time etc.	
availabilityTime (eo:timePosition)	The time when the result became available (i.e. updated) in the catalogue.	
acquisitionStation (eo:acquisitionStation)	The station used for the acquisition.	
acquisitionSubType (eo:acquisitionSubType)	The Acquisition sub-type.	
startTimeFromAscendingNode (eo:startTimeFromAscendingNode)	Start time of acquisition in milliseconds from Ascending node date.	
completionTimeFromAscendingNode (eo:completionTimeFromAscendingNode)	Completion time of acquisition in milliseconds from Ascending node date.	
illuminationAzimuthAngle (eo:illuminationAzimuthAngle)	Mean illumination/solar azimuth angle given in degrees.	
illuminationZenithAngle (eo:illuminationZenithAngle)	Mean illumination/solar zenith angle given in degrees.	
illuminationElevationAngle (eo:illuminationElevationAngle)	Mean illumination/solar elevation angle given in degrees.	
polarisationMode (eo:polarisationMode)	The polarisation mode taken from codelist: S (for single), D (for dual), T (for twin), Q (for quad), UNDEFINED	polarisationMode=D
polarisationChannels (eo:polarisationChannels)	Polarisation channel transmit/receive configuration.	polarisationChannels=vertical
antennaLookDirection (eo:antennaLookDirection)	LEFT or RIGHT.	
minimumIncidenceAngle (eo:minimumIncidenceAngle)	Minimum incidence angle given in degrees.	
maximumIncidenceAngle (eo:maximumIncidenceAngle)	Maximum incidence angle given in degrees.	
dopplerFrequency (eo:dopplerFrequency)	Doppler Frequency of acquisition.	
incidenceAngleVariation (eo:incidenceAngleVariation)	Incidence angle variation	

Parameters

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

[35] http://www.doi.org/

Table 17: 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 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[36].	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:	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 GEOS[37], 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.

1.8.6 The Webclient Interface

Table of Contents

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[38] display to show a WMS view of the datasets within a map context. The background map tiles are provided by EOX[39].

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`.

[36] https://en.wikipedia.org/wiki/ISO_8601
[37] https://trac.osgeo.org/geos/ticket/377
[38] http://openlayers.org/
[39] https://maps.eox.at//

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 command[40].

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`.

```
from django.urls import include, re_path

urlpatterns = [
    ...
    re_path(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* wouldopen 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.

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.

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, subcetted by the given spatial subsets.

The "About" panel shows general info of EOxClient[41], 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. Hollow

[40] https://docs.djangoproject.com/en/1.8/ref/contrib/staticfiles/#collectstatic
[41] https://github.com/EOX-A/EOxClient

Fig. 1: *The webclient showing the contents of the autotest instance.*

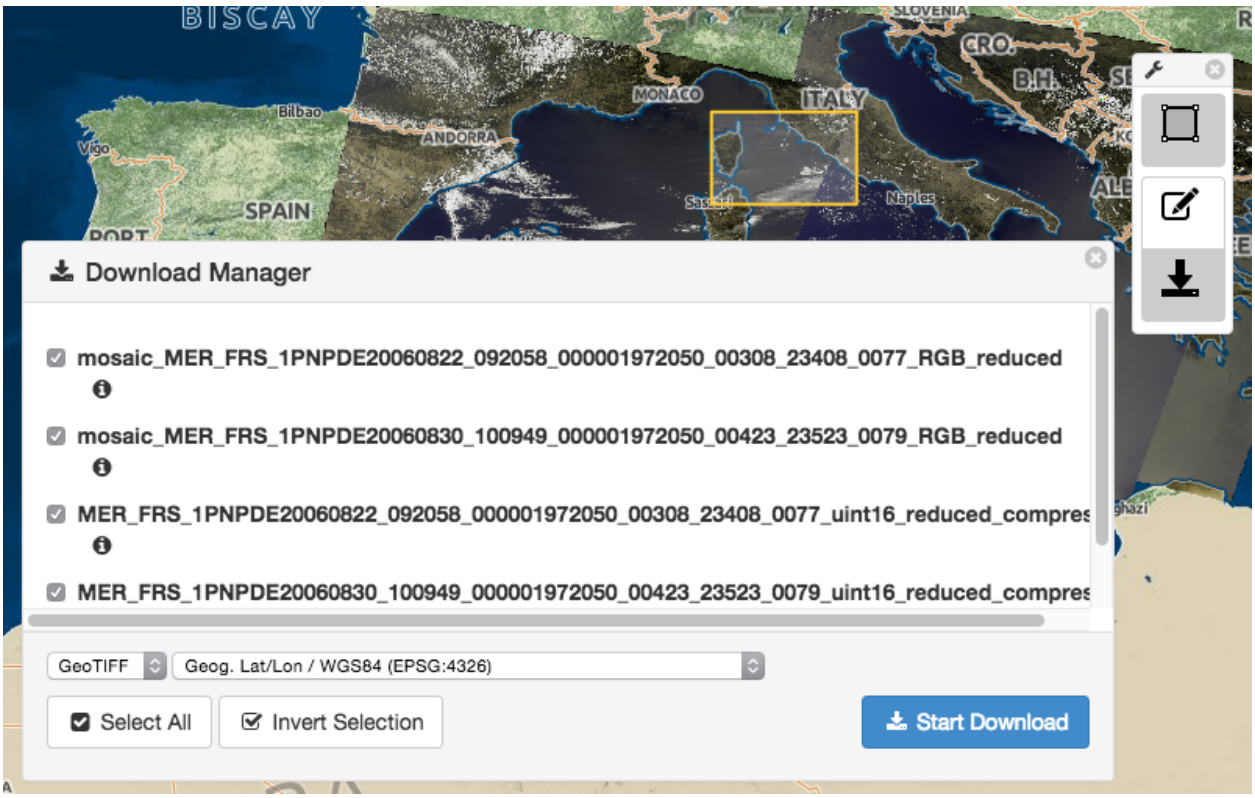


Fig. 2: *The download selection view.*

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.

1.8.7 Common Query Language (CQL)

This document describes the basic syntax of the common query language. CQL is the query language defined the Catalogue Service specification (CSW)[42]. CQL support in EOxServer is realized using the external pycql[43] package.

This document is based upon the (E)CQL documentation of GeoServer[44] with adapations wherever needed.

Syntax Specification

This chapter shows the syntax to define CQL queries.

[42] http://docs.opengeospatial.org/is/12-168r6/12-168r6.html
[43] https://pycql.readthedocs.io/
[44] https://docs.geoserver.org/latest/en/user/filter/ecql_reference.html

Condition

Table 18: condition syntax

Syntax	Description
Predicate (page 55)	A single predicate expression
Condition (page 55) `AND` | `OR` *Condition* (page 55)	Logical combination of two conditions
`NOT` *Condition* (page 55)	Logical inversion of a condition.
`(` | `[` *Condition* (page 55) `]` | `)`	Grouping of conditions regarding evaluation order

Predicate

Table 19: predicate syntax

Syntax	Description
Expression (page 56) = | <> | < | <= | > | >= *Expression* (page 56)	Comparison of two expression
Expression (page 56) [`NOT`] `BETWEEN` *Expression* (page 56) `AND` *Expression* (page 56)	Value range Comparison
Expression (page 56) [`NOT`] `LIKE` | `ILIKE` `pattern`	Check whether an expression matches a pattern. The `%` character can be used as a wildcard.
Expression (page 56) [`NOT`] `IN` `(` *Expression* (page 56) { `,` *Expression* (page 56) `)`	Tests the inclusion of a value in a set of values.
Expression (page 56) `IS` [`NOT`] `NULL`	Tests whether the evaluated expression is `NULL`
Expression (page 56) `BEFORE` *Timestamp* (page 56)	
Expression (page 56) `BEFORE OR DURING` *Period* (page 56)	
Expression (page 56) `DURING` *Period* (page 56)	
Expression (page 56) `DURING OR AFTER` *Period* (page 56)	
Expression (page 56) `AFTER` *Timestamp* (page 56)	
`INTERSECTS` `(` *Expression* (page 56) `,` *Expression* (page 56) `)`	
`DISJOINT` `(` *Expression* (page 56) `,` *Expression* (page 56) `)`	
`CONTAINS` `(` *Expression* (page 56) `,` *Expression* (page 56) `)`	
`WITHIN` `(` *Expression* (page 56) `,` *Expression* (page 56) `)`	
`TOUCHES` `(` *Expression* (page 56) `,` *Expression* (page 56) `)`	
`CROSSES` `(` *Expression* (page 56) `,` *Expression* (page 56) `)`	
`OVERLAPS` `(` *Expression* (page 56) `,` *Expression* (page 56) `)`	
`EQUALS` `(` *Expression* (page 56) `,` *Expression* (page 56) `)`	
`RELATE` `(` *Expression* (page 56) `,` *Expression* (page 56) `,` pattern `)`	
`DWITHIN` `(` *Expression* (page 56) `,` *Expression* (page 56) `,` *Number* (page 56) `,` units `)`	
`BEYOND` `(` *Expression* (page 56) `,` *Expression* (page 56) `,` *Number* (page 56) `,` units `)`	
`BBOX` `(` *Expression* (page 56) `,` *Number* (page 56) `,` *Number* (page 56) `,` *Number* (page 56) `,` *Number* (page 56) [`,` CRS] `)`	

Expression

Table 20: expression syntax

Syntax	Description
Attribute (page 56)	Name of an objects attribute
Literal (page 56)	A literal value
Expression (page 56) + | − | ∗ | / *Expression* (page 56)	Arithmetic operations of two expressions
(| [*Expression* (page 56)] |)	Grouping of expression regarding evaluation order

Literal

Table 21: literal values

Syntax	Description
Number	A literal number (either floating point or integer
Boolean	A literal booelan value: either *TRUE* or *FALSE*
Timestamp	A timestamp literal. Must be in ISO 8601 compliant datetime format.
Duration	A timestamp literal. Must be in ISO 8601 compliant duration format.
Geometry	A Geometry in WKT format. EPSG:4326 is assumed

Period

Table 22: period syntax

Syntax	Description
Timestamp (page 56) / *Timestamp* (page 56)	Period definition using the start and end timestamp.
Timestamp (page 56) / *Duration* (page 56)	Period definition using the start timestamp and a duration afterwards.
Duration (page 56) / *Timestamp* (page 56)	Period definition using the end timestamp and a duration before.

Attribute

Depending on the current query context, the following attributes are available to use in the queries.

Table 23: available attributes

Attribute name	Field type	Availability
identifier	String	All
beginTime	Timestamp	All
endTime	Timestamp	All
footprint	String	All
inserted	Timestamp	All
updated	Timestamp	All
productType	String	Collection

Continued on next page

Table 23 – continued from previous page

Attribute name	Field type	Availability
doi	String	Collection
platform	String	Collection
platformSerialIdentifier	String	Collection
instrument	String	Collection
sensorType	String	Collection
compositeType	String	Collection
processingLevel	String	Collection
orbitType	String	Collection
spectralRange	String	Collection
wavelength	Number	Collection
parentIdentifier	String	Product
productionStatus	String	Product
acquisitionType	String	Product
orbitNumber	Number	Product
orbitDirection	Number	Product
track	Number	Product
frame	Number	Product
swathIdentifier	String	Product
productVersion	String	Product
productQualityStatus	String	Product
productQualityDegradationTag	String	Product
processorName	String	Product
processingCenter	String	Product
creationDate	Timestamp	Product
modificationDate	Timestamp	Product
processingDate	Timestamp	Product
sensorMode	String	Product
archivingCenter	String	Product
processingMode	String	Product
availabilityTime	Timestamp	Product
acquisitionStation	String	Product
acquisitionSubType	String	Product
startTimeFromAscendingNode	Number	Product
completionTimeFromAscendingNode	Number	Product
illuminationAzimuthAngle	Number	Product
illuminationZenithAngle	Number	Product
illuminationElevationAngle	Number	Product
polarisationMode	String	Product
polarizationChannels	String	Product
antennaLookDirection	String	Product
minimumIncidenceAngle	Number	Product
maximuIncidenceAngle	Number	Product
dopplerFrequency	Number	Product
incidenceAngleVariation	Number	Product
cloudCover	Number	Product
snowCover	Number	Product
lowestLocation	Number	Product
highestLocation	Number	Product

1.9 Operations Guide

This guide helps with the setup, configuration and management of an operational deployment of EOxServer.

1.9.1 Recommendations for Operational Installation

Table of Contents

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 5)** generic installation procedure for GNU/Linux operating systems.
- ***Installation on CentOS*** **(page 65)** for specific installation on CentOS.
- ***Creation*** **(page 7)** 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 5) and *CentOS* (page 65) installation guides) the *configuration* (page 7) 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[45]:

```
/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).

[45] http://www.pathname.com/fhs/pub/fhs-2.3.html#SRVDATAFORSERVICESPROVIDEDBYSYSTEM

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 by 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 EOxSerevr 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.

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 PosgreSQL are covered, e.g., here[46].

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.. 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 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.

[46] http://www.postgresql.org/docs/devel/static/auth-pg-hba-conf.html

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 66)) or via the Python Package Index (see *Alternate installation method using pip* (page 67)).

This guide assume that the sudo[47] 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 Apache[48] web server using the WSGI[49] 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).

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

:: 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 **iptable** firewall settings permanent (preserved throughout reboots) run:

:: sudo service iptables save

[47] http://www.centos.org/docs/4/4.5/Security_Guide/s3-wstation-privileges-limitroot-sudo.html
[48] http://www.apache.org/
[49] http://en.wikipedia.org/wiki/Web_Server_Gateway_Interface

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 PostgreSQL[50] with PostGIS[51] 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., PosgreSQL's wiki[52]

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 deamon 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
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[53] command.

[50] http://www.postgresql.org/
[51] http://postgis.net/
[52] http://wiki.postgresql.org/wiki/Main_Page
[53] http://unixhelp.ed.ac.uk/CGI/man-cgi?useradd+8

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 Postgress DBMS is up an 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
```

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 PosgreSQL documentation[54].

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.:

[54] http://www.postgresql.org/docs/devel/static/auth-pg-hba-conf.html

```
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 empry 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

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:

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[55]):

[55] http://code.google.com/p/modwsgi/wiki/ConfigurationIssues

:: 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:

```
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.

1.9.2 Installation on CentOS

Table of Contents

This section describes specific installation procedure for EOxServer on CentOS[56] 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 5)** generic installation procedure for GNU/Linux operating systems.
- *Creation* **(page 7)** to configure an instance of EOxServer after successful installation.
- *Recommendations for Operational Installation* **(page 58)** 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)[57] repository which can be added with the following *yum* command:

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

The second repository to be added is EPEL (Extra Packages for Enterprise Linux)[58] again via a simple *yum* command:

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

Finally EOxServer is available from the yum repository at packages.eox.at[59]. This repository offers current versions of packages like MapServer[60] as well as custom built ones with extra drivers enabled like GDAL[61] and/or with patches applied like libxml2[62]. 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
```

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

[56] http://www.centos.org/
[57] http://wiki.osgeo.org/wiki/Enterprise_Linux_GIS
[58] http://fedoraproject.org/wiki/EPEL
[59] http://packages.eox.at
[60] http://mapserver.org/
[61] http://gdal.org/
[62] http://xmlsoft.org/

```
sudo yum update
```

Please carefully follow the migration/update procedure 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 PosgreSQL 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 7).

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[63] 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 7).

[63] http://www.pip-installer.org/

Special *pysqlite* considerations

When used with spatialite[64] EOxServer also requires pysqlite[65] and *pyspatialite* which can be either installed as RPMs from packages.eox.at[66] (see *Installing EOxServer* (page 66) above) or from source.

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
```

1.9.3 Management

This chapter deals with the operational management of an EOxServer instance. It is assumed, that EOxServer is installed, an instance is created and configured. For more information please refer to the *Installation* (page 5), *Creation* (page 7), and *Configuration* (page 7) sections respectively. Also, data preprocessing is not part of the this guide.

This guide will use a practical example of real high resolution RGB + near infrared satellite imagery from the SPOT mission to show how to set up an operational service. To add a little more complexity, the data type is 16 bit unsigned integer, which is common for many earth observation instruments.

Setup

Each instance will most likely deal with a limited set of data and semantics, thus it is beneficial to provide a strict configuration of the underlying types in order to improve coherence, add metadata and ensure integrity.

For our example we start with the lowest level of abstractions, the coverages. As the data to be ingested consists of RGB + NIR files, the used coverage type needs to reflect just that.

The following JSON definition is used to specify the fields of the coverage type and to provide some extra metadata. The contents are stored in the file `rgbnir.json`:

```
{
    "bands": [
        {
            "definition": "http://www.opengis.net/def/property/OGC/0/Radiance",
            "description": "Red Channel",
            "gdal_interpretation": "RedBand",
            "identifier": "red",
            "name": "red",
            "nil_values": [
```

(continues on next page)

[64] http://www.gaia-gis.it/spatialite/
[65] http://code.google.com/p/pysqlite/
[66] http://packages.eox.at

(continued from previous page)

```
                {
                    "reason": "http://www.opengis.net/def/nil/OGC/0/unknown",
                    "value": 0
                }
            ],
            "uom": "W.m-2.Sr-1",
            "significant_figures": 5,
            "allowed_value_ranges": [
                [0, 65535]
            ]
        },
        {
            "definition": "http://www.opengis.net/def/property/OGC/0/Radiance",
            "description": "Green Channel",
            "gdal_interpretation": "GreenBand",
            "identifier": "green",
            "name": "green",
            "nil_values": [
                {
                    "reason": "http://www.opengis.net/def/nil/OGC/0/unknown",
                    "value": 0
                }
            ],
            "uom": "W.m-2.Sr-1",
            "significant_figures": 5,
            "allowed_value_ranges": [
                [0, 65535]
            ]
        },
        {
            "definition": "http://www.opengis.net/def/property/OGC/0/Radiance",
            "description": "Blue Channel",
            "gdal_interpretation": "BlueBand",
            "identifier": "blue",
            "name": "blue",
            "nil_values": [
                {
                    "reason": "http://www.opengis.net/def/nil/OGC/0/unknown",
                    "value": 0
                }
            ],
            "uom": "W.m-2.Sr-1",
            "significant_figures": 5,
            "allowed_value_ranges": [
                [0, 65535]
            ]
        },
        {
            "definition": "http://www.opengis.net/def/property/OGC/0/Radiance",
            "description": "Nir Channel",
            "gdal_interpretation": "NirBand",
            "identifier": "nir",
            "name": "nir",
            "nil_values": [
                {
                    "reason": "http://www.opengis.net/def/nil/OGC/0/unknown",
                    "value": 0
```

(continues on next page)

(continued from previous page)

```
                }
            ],
            "uom": "W.m-2.Sr-1",
            "significant_figures": 5,
            "allowed_value_ranges": [
                [0, 65535]
            ]
        }
    ],
    "data_type": "Uint16",
    "name": "RGBNir"
}
```

This definition can now be loaded in the services using the `coveragetype import` command:

```
python manage.py coveragetype loaddata rgbnir.json
```

Now that the Coverage type is registered, it can be used to create one or multiple Product types. This takes the rather abstract Coverage type and creates a more specific type structure data for a certain satellite mission or instrument. The following command creates such a product type for `PL00` Products, referencing the previously imported Coverage type `RGBNir`.

```
python manage.py producttype create PL00 --coverage-type RGBNir
```

For the generated Product type, we can now add visual representations, called Browse types in EOxServer. Browse types can be defined to create definitions for RGB, RGBA or color scaled images from the registered coverages. This is achieved by providing transfer functions using either the band names or expressions and additional value ranges and no-data values.

For the example, three Browse types are created: true color RGB, false color RGB, and a grayscale NDVI using the red and near infrared bands. The following commands will do just that, plus creating a fourth Browse type (a copy of the `TRUE_COLOR` one) with no name, marking it as the default representation.

```
python manage.py browsetype create PL00 \
    --red "red" \
    --green "green" \
    --blue "blue" \
    --red-range 1000 15000 \
    --green-range 1000 15000 \
    --blue-range 1000 15000 \
    --red-nodata 0 \
    --green-nodata 0 \
    --blue-nodata 0

python manage.py browsetype create PL00 TRUE_COLOR \
    --red "red" \
    --green "green" \
    --blue "blue" \
    --red-range 1000 15000 \
    --green-range 1000 15000 \
    --blue-range 1000 15000 \
    --red-nodata 0 \
    --green-nodata 0 \
    --blue-nodata 0

python manage.py browsetype create PL00 FALSE_COLOR \
```

(continues on next page)

(continued from previous page)

```
    --red "nir" \
    --green "red" \
    --blue "green" \
    --red-range 1000 15000 \
    --green-range 1000 15000 \
    --blue-range 1000 15000 \
    --red-nodata 0 \
    --green-nodata 0 \
    --blue-nodata 0

python manage.py browsetype create PL00 NDVI \
    --grey "(nir-red)/(nir+red)" --grey-range -1 1
```

For true and false color representations, a red, green, and blue band is selected using the names as defined in the `RGBNir` range type. Using the `range` selectors the input range is specified which will be linearly scaled to produce a normalized value range of the output image. The nodata values help to mark out pixels that ought to be transparent.

The `NDVI` Browse type uses the `--grey` output band with a mathematical expression. The variables names in the expression must use the band names of the Coverage type. Using the `--grey-range`, a default value range is specified.

It is typical that EO data products entail vector masks to mark areas with a specific property. Usually this is used to mark the (in-)validity in a specific region or to mark clouds or snow.

In order to take advantage of these masks, for each type of mask a Mask type must be registered. In our example, only the single validity mask is used. To "mask-in" areas the specific `--validity` flag must be used, otherwise the inverse is assumed.

```
python manage.py masktype create --validity PL00 validity
```

Note: It is possible to combine the data of multiple Product types. In those cases it is important to define the same Browse and Mask types (even if the underlying expressions/ranges/no-data values are different), so that they can be rendered as a single map layer.

The final step in the setup of the types is to create a Collection type. It is possible to put both Coverages and Products into a collection, so it is a good practice to limit the types of Products and Coverages that can be added to what is actually required.

The following Collection type creation command specifies that it is possible to put both Coverages and Products of the previously created types into such a Collection.

```
python manage.py collectiontype create CollectionType \
    --coverage-type RGBNir \
    --product-type PL00
```

Since we will most likely have only one or a very limited amount of Collections in the lifetime of the service, the instantiation of the Collection could be considered as part of the setup procedure.

```
python manage.py collection create Collection --type CollectionType
```

One task that must be prepared when using more sophisticated storage mechanisms is to specify the Storage backends and their respective Storage authentication/authorization mechanisms. For our example, we assume that our data resides on an OpenStack Swift object storage. This requires a keystone authentication system which can be set up in the following manner (auth credentials are assumed to be in the used bash environment variables):

```
python manage.py storageauth create auth-keystone https://auth.obs.service.com \
    --type keystone \
    -p auth-version "${ST_AUTH_VERSION}" \
    -p identity-api-version="${ST_AUTH_VERSION}" \
    -p username "${OS_USERNAME}" \
    -p password "${OS_PASSWORD}" \
    -p tenant-name "${OS_TENANT_NAME}" \
    -p tenant-id "${OS_TENANT_ID}" \
    -p region-name "${OS_REGION_NAME}"
```

We can now create a named Storage of the type `swift` using the keystone auth object from above:

```
python manage.py storage create \
    my-storage ${CONTAINER} \
    --type swift \
    --storage-auth auth-keystone
```

This concludes the setup step and the service is now ready to be ingested with data.

Data registration

Products and Coverages can be ingested using the command line interface as well.

In our example, we assume that our data files are structured in the following way:

- all files reside on a Swift object storage, the one established in the *Setup* (page 68) section.
- all acquisitions are stored as ZIP containers, which include the raster data, vector masks and metadata in GSC format.
- the raster data are comprised of one TIFF file per band, one each for red, green, blue, and near infrared with their file suffix indicating their semantics.

The first step is to register the Product itself. This is done by referencing the ZIP container itself.

```
product_identifier=$(
    python manage.py product register \
        --type PL00 \
        --collection Collection \
        --meta-data my-storage path/to/package.zip metadata.gsc \
        --package my-storage path/to/package.zip \
        --print-identifier
)
```

The management command prints the identifier of the registered coverage, which is stored in a bash variable. It can be used to associated the Coverages to the product. Using the `--collection` parameter, the Product is automatically put into the Collection created earlier.

The next step is to register a Coverage and associate it with the Product.

```
python manage.py coverage register \
    --type RGBNir \
    --product ${product_identifier} \
    --identifier "${product_identifier}_coverage" \
    --meta-data my-storage path/to/package.zip metadata.gsc \
    --data my-storage path/to/package.zip red.tif \
    --data my-storage path/to/package.zip green.tif \
```

(continues on next page)

(continued from previous page)

```
    --data my-storage path/to/package.zip blue.tif \
    --data my-storage path/to/package.zip nir.tif
```

For the data access let us define that the Product identifier is `Product-A` this the Coverages identifier is `Product-A_coverage`.

Data access

Now that the first Product and its Coverage are successfully registered, the services can already be used.

Web Map Service (WMS)

Via WMS it is possible to get rendered maps from the stored Products and Coverages. The table for Layer Mapping is imporant here. From that we can deduct various map layers that are available for access.

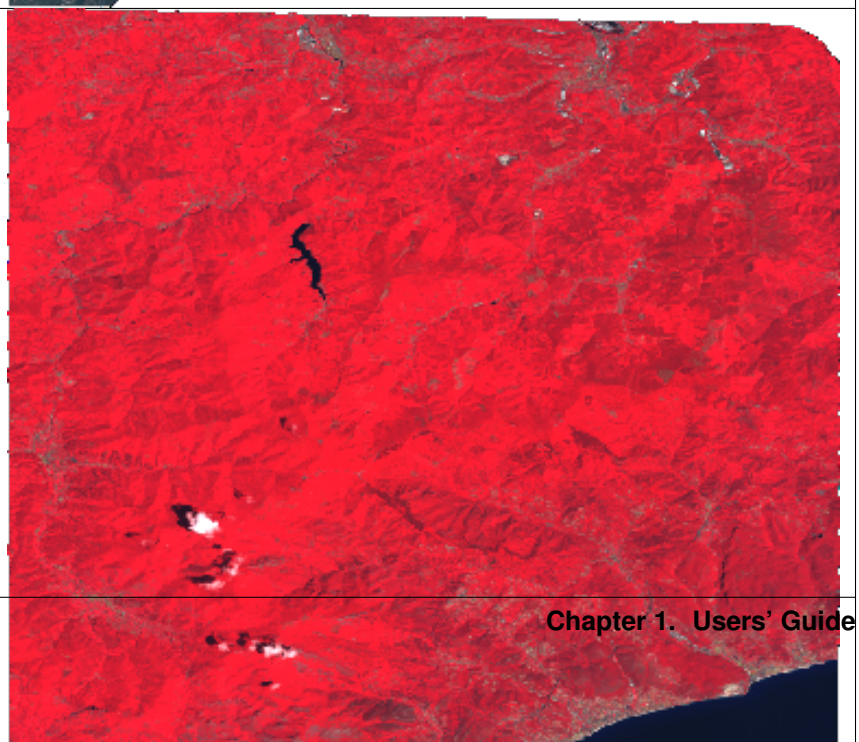
For production services it is typical to provide access to thounsands of earth observation Products, thus rendering individual Product access impractical for visual browsing. Typically, it is more convenient to access the Collection instead using the area and time of interest and optionally additional metadata filters.

This results in a catalog of the following available layers:

- `Collection`: the most basic rendering of the Collection. In our example the we created four Browse Type definitions: `TRUE_COLOR`, `FALSE_COLOR`, `NDVI` and an unnamed default one which had the same parameters as `TRUE_COLOR`. This means, that the default rendering is a true color representation of the Products.
- `Collection__outlines`: this renders the outlines of the Products as geometries.
- `Collection__outlined`: this is a combination of the previous two layers: each Product is rendered in `TRUE_COLOR` with its outlines highlighted.
- `Collection__TRUE_COLOR`, `Collection__FALSE_COLOR`, `Collection__NDVI`: these are the browse visualizations with the definintions from earlier.
- `Collection__validity`: this renders the Products vector masks as colored geometries.
- `Collection__masked_validity`: this renders the default visualization (true color) but applies each Products validity mask.

The following list shows all of these rendering options with an example product

Table 24: WMS Collection Layers

Layer	Example image
`Collection/ Collection__TRUE_COLOR`	
`Collection__FALSE_COLOR`	

It is possible to filter the objects using their metadata. This happens already with the mandatory `bbox`: only objects that intersect with that bounding box are further processed and rendered to the output map. One other such parameter is the `time` parameter. It allows to specify a time instant or a time range to include objects.

It is, however, also possible to filter upon any other metadata of a Product as well. This can be used, for example, to only render images below a threshold of cloud coverage, to generate a mosaic of almost cloud free images. The parameter to use is the `cql` one. For our example, we would append `&cql=cloudCover <= 5` to only include images with less or equal than 5% cloud coverage. For this to work, the metadata of the Products needs to be indexed upon registration. This is done in the process of metadata reading.

For more details about CQL and all available metadata fields refer to the *Common Query Language (CQL)* (page 54) documentation.

Web Coverage Service (WCS)

WCS in EOxServer uses a more straight-forward mapping of EO object types to WCS data model types. As EOxServer makes use of the EO Application Profile it maps Mosaics and Coverages to Rectified Stitched Mosaics and Rectified/Referenceable Datasets respectively and Collections and Products to Dataset Series.

Table 25: WCS EO Object type mapping

Object type	EO-WCS data model type
Coverage	Rectified Dataset/Referenceable Dataset (depending on whether or not a Grid is used).
Product	DatasetSeries
Mosaic	RectifiedStitchedMosaic
Collection	DatasetSeries

For our example this means that a typical client will fist investigate the WCS capabilities document to find out what Dataset Series are available, as listing a very large amount of Coverages is not feasible. In our example, the `Collection` is listed as Dataset Series.

To explore it further, `DescribeEOCoverageSet` request with spatio-temporal subsets can be used to get the contents of the Dataset Series. This will list the entailed Products as sub Dataset Series and the Coverages as their respective EO Coverage type.

All Coverages of interest can be downloaded using `GetCoverage` requests.

OpenSearch

The access to the indexed objects via OpenSearch uses the two-step search principle: the root URL of OpenSearch returns with the general OpenSearch description document (OSDD), detailing the available search patterns using URL templates. Each template is associated with a result format in which the search results are rendered. The first step is to search for advertised Collections.

For our example, this will return our single `Collection` encoded in the chosen result format. This also includes

Table 26: OpenSearch URL endpoints

URL	Semantic
`opensearch`	The root OSDD file.
`opensearch/<format>`	The collection search step
`opensearch/<format>`	The search for collections using the specified format
`opensearch/collections/Collection`	The OSDD file specific to the `Collection`
`opensearch/collections/Collection/<format>`	The search for items in our `Collection` in that format

CHAPTER 2

Developers' Guide

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.

2.1 Basics

Table of Contents

This is a short description of the basic elements of the EOxServer software architecture.

2.1.1 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 /users/install 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[67] which EOxServer uses through its Python bindings to handle certain OGC Web Service requests.

GDAL/OGR

In some cases EOxServer uses the GDAL/OGR[68] library for access to geospatial data directly (rather than through MapServer).

2.2 Core

2.3 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 79)".

2.3.1 Data Integration Layer

Figure: "*EOxServer Data Model for Coverage Resources* (page 79)" below shows the data model of the coverage resources. Note the correlation with the EO-WCS data model as shown above.

2.3.2 Data Access Layer

Figure: "*EOxServer Data Model for Back-ends* (page 80)" below shows the data model of the back-ends layer.

[67] http://www.mapserver.org
[68] http://www.gdal.org

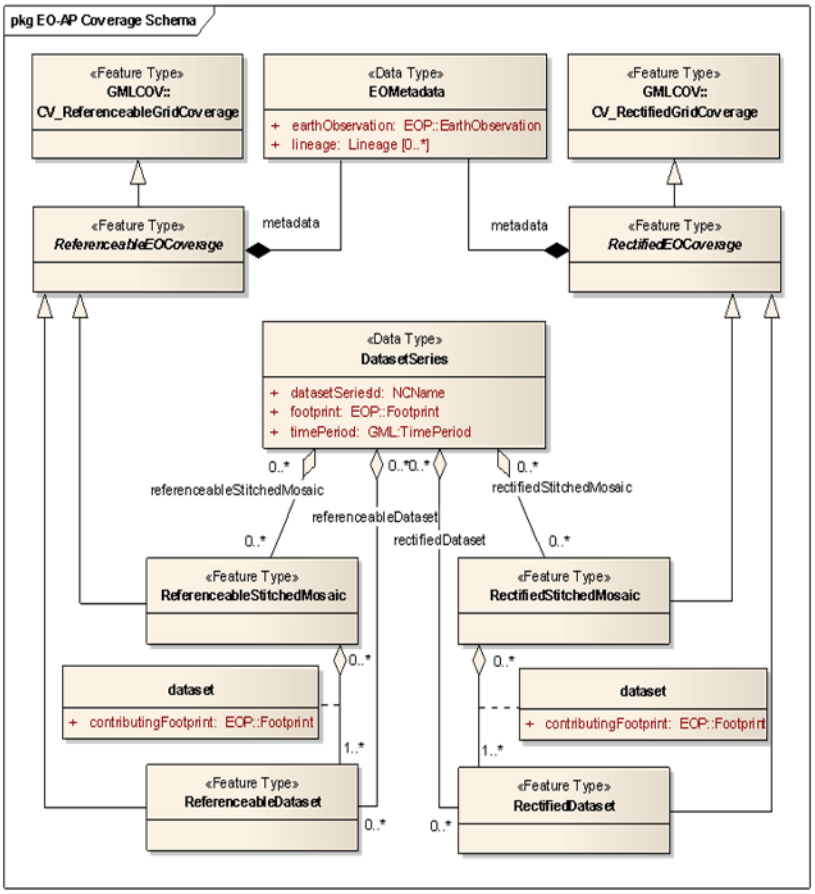


Fig. 1: *EO-WCS Data Model from OGC 10-140*

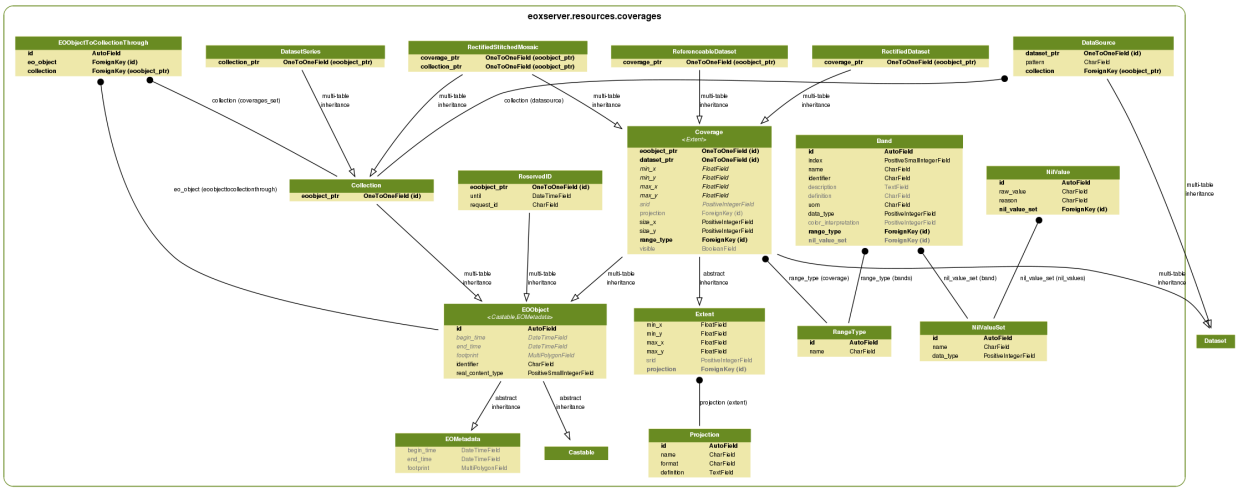


Fig. 2: *EOxServer Data Model for Coverage Resources*

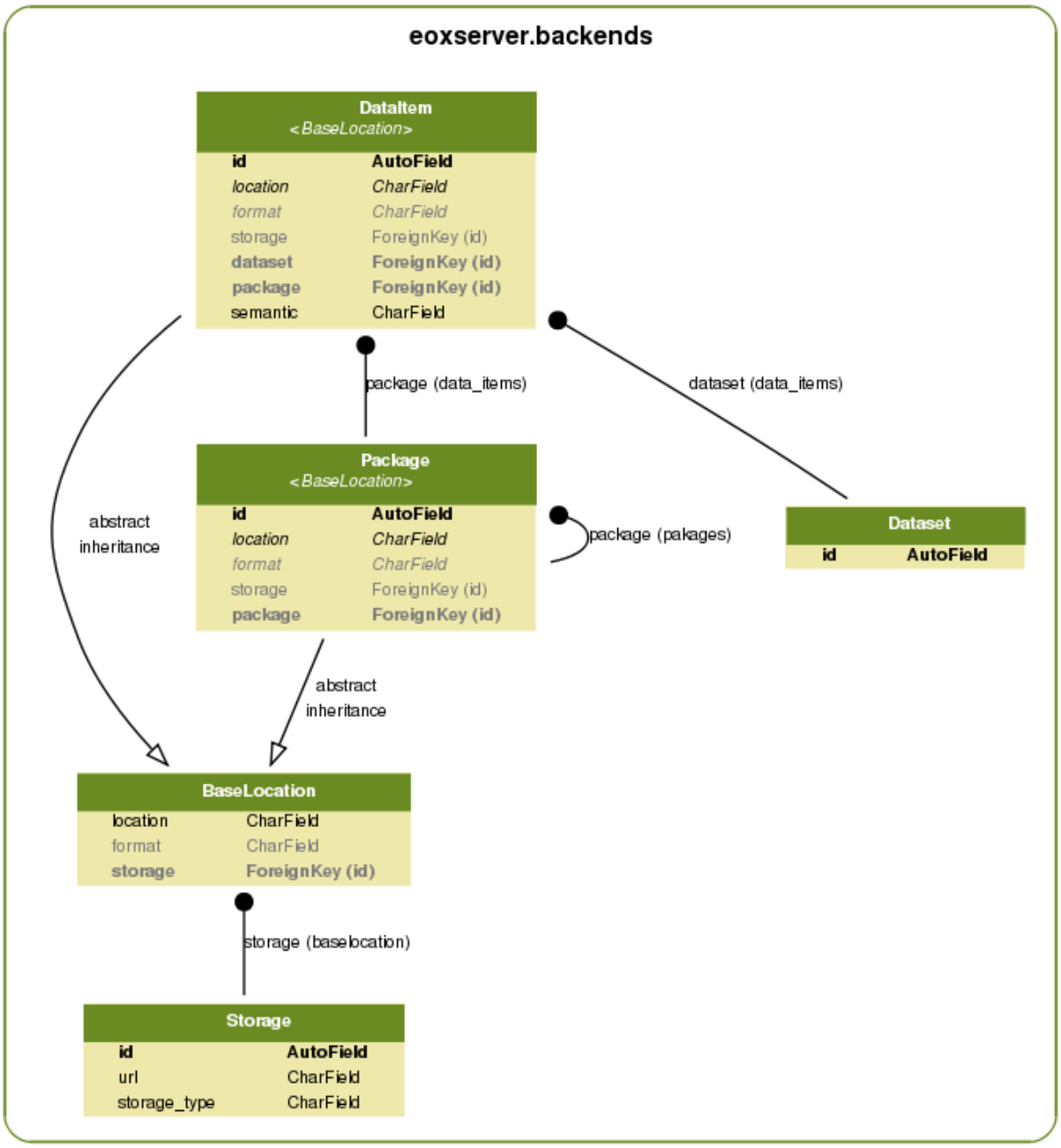


Fig. 3: *EOxServer Data Model for Back-ends*

2.3.3 Task Tracker Data Model

Asynchrounous Task Processing (ATP) uses its own DB model displayed in Figure: "*EOxServer Data Model of ATP Task Tracker* (page 81)" to implement the task queueu, store the task inputs and outputs and track the tasks' status. (For more detail on ATP subsystem see "atp_sum").

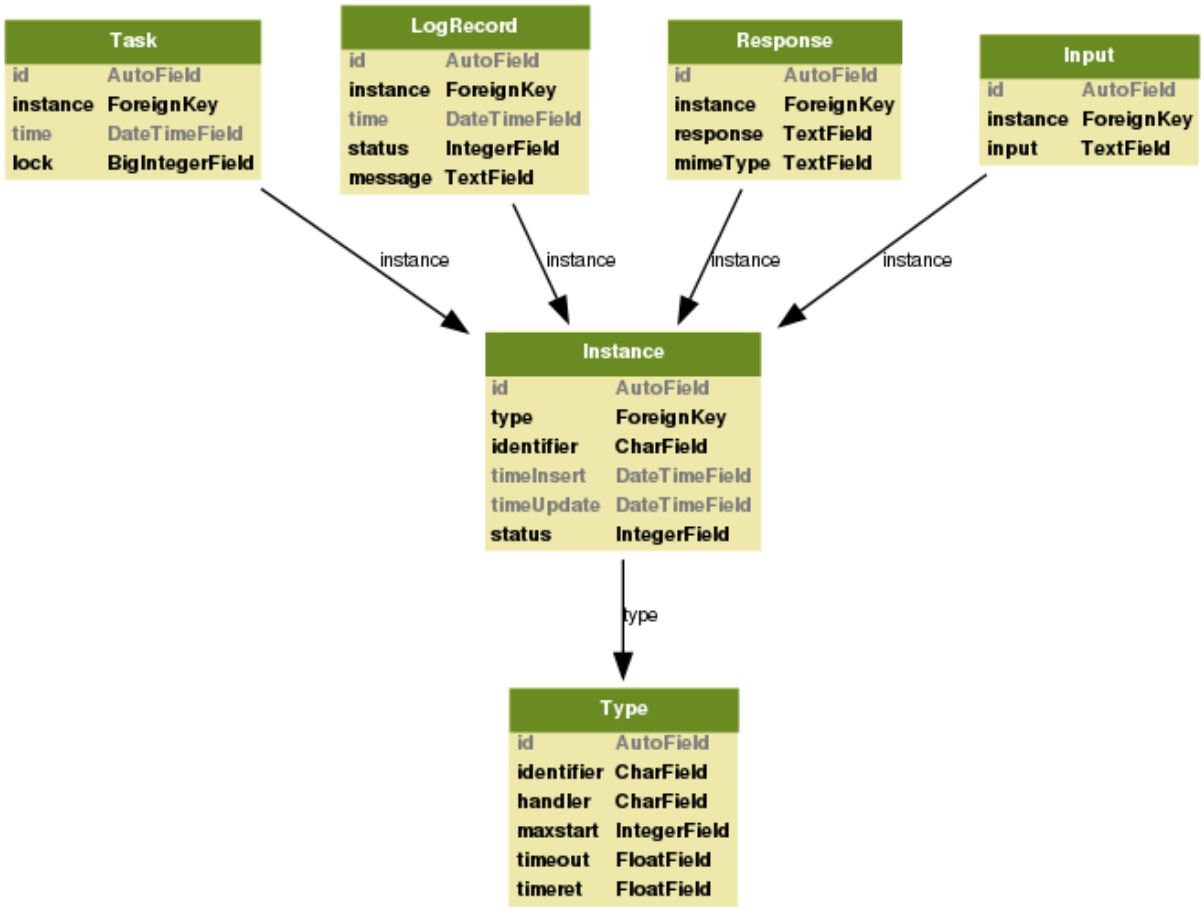


Fig. 4: *EOxServer Data Model of ATP Task Tracker*

2.4 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[69] for schema and data migration management.

2.4.1 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

[69] http://south.aeracode.org/

> 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 documentation[70]

2.4.2 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[71].

2.4.3 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 *Souths* 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[72].

2.4.4 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*).

2.5 Plugins

EOxServer uses a plugin framework to extend or alter the built-in functionality. The plugin system is based on trac's Component Architecture[73]. We copied the relevant file as *eoxserver.core.component* (page 128) to not add the full trac framework as a dependency.

EOxServer plugins are classes that inherit from *eoxserver.core.component.Component* (page 128). 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* (page 128).

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.

[70] http://south.readthedocs.org/en/latest/whataremigrations.html
[71] http://south.readthedocs.org/en/latest/installation.html
[72] http://south.readthedocs.org/en/latest/tutorial/part1.html
[73] http://trac.edgewall.org/wiki/TracDev/ComponentArchitecture

2.5.1 Important

Components should not be created manually, but only be retrieved via an *eoxserver.core.component.ExtensionPoint* (page 128). 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.

2.5.2 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 191) 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.

2.5.3 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.

2.6 Services

This section deals with the creation of new Hervices handlers that allow to process OGC web service requests and are easily exposed via the *ows* (page 196) view.

Service Handlers are *Components* (page 128) that at least implement the *ServiceHandlerInterface* (page 190). For a Service Handler to be fully accessible it is also necessary to implement either or both of *GetServiceHandlerInterface* (page 190) and *PostServiceHandlerInterface* (page 190). For general information about Plugins/Components please refer to the *Plugins* (page 82) documentation.

2.6.1 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`[74] 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):
```

(continues on next page)

[74] https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.HttpRequest

(continued from previous page)

```
    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".

2.7 Data Formats

2.8 Metadata Formats

2.9 The *autotest* instance

Table of Contents

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

2.9.1 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 86). 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.

2.9.2 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
```

2.9.3 Deployment

The autotest instance can be deployed like any other EOxServer instance. The fastest way to actually access the data just run:

```
python manage.py runserver 0.0.0.0:8000
```

2.9.4 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"
```

2.10 SOAP Proxy

Table of Contents

2.10.1 Architecture

Soap_proxy is an adapter proxy which accepts POST request in XML ecoded 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.

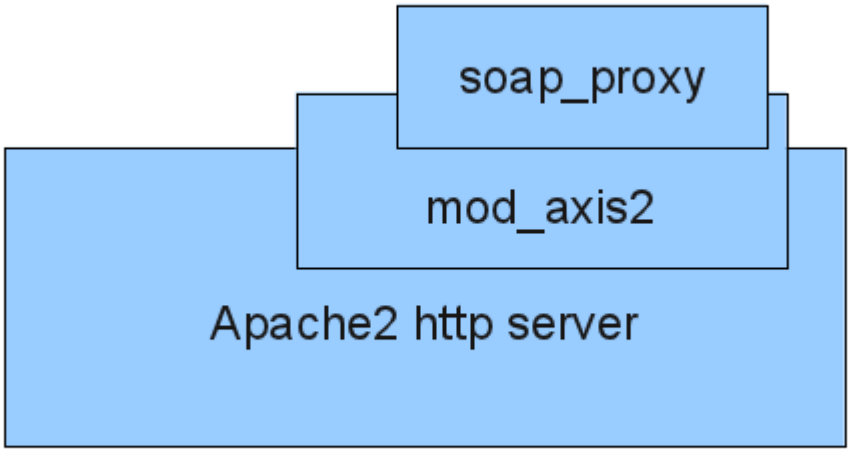
EOxServer responds to the following WCS-EO requests through SOAP service interface:

- 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.

2.10.2 Implementation

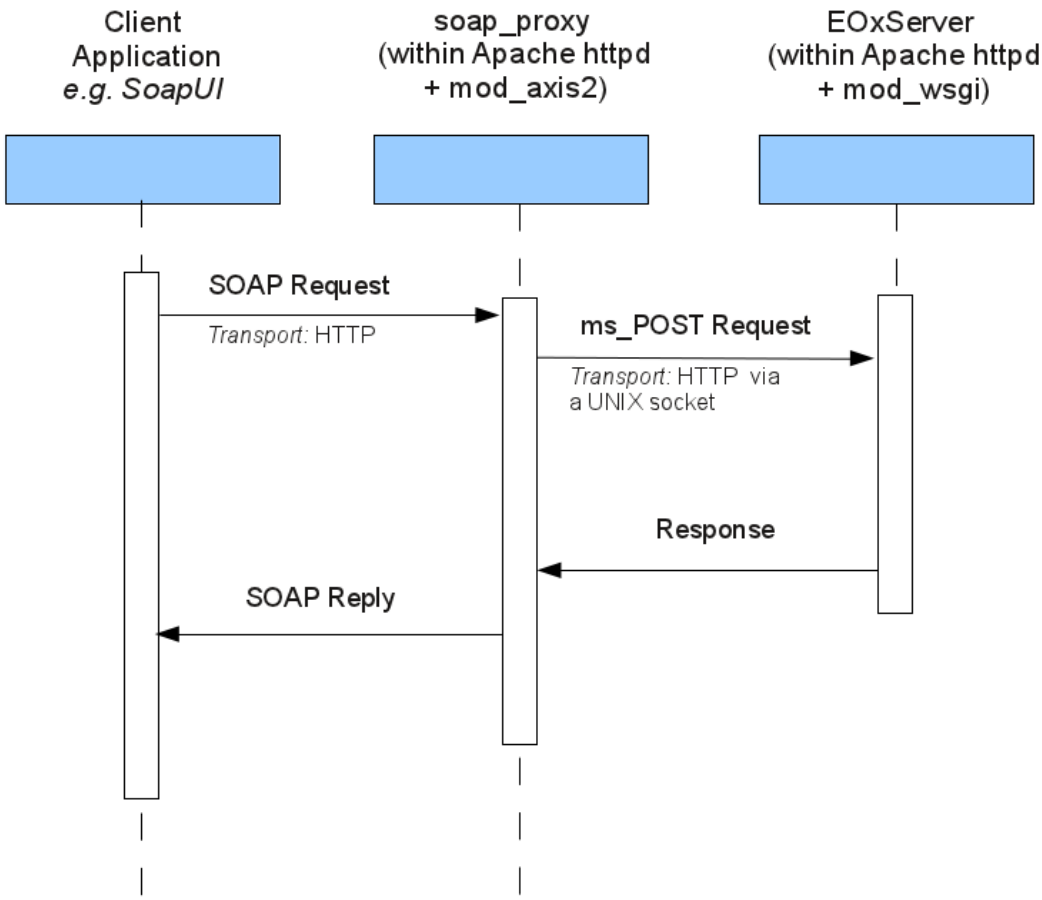
The implementation is provided in the src directory. The file sp_svc.c is the entry point where the Axis2/c framework calls the soap_proxy implentation code via rpSvc_invoke(), which calls rp_dispatch_op() to do most of the work.

2.11 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`[75] class all associated documentation is also applicable to all EOxServer models. Highly recommendable is also the Django QuerySet documentation[76],

[75] https://docs.djangoproject.com/en/2.2/ref/models/instances/#django.db.models.Model
[76] https://docs.djangoproject.com/en/dev/ref/models/querysets/


Client
Application
e.g. SoapUI
soap_proxy
(within Apache httpd
+ mod_axis2)
EOxServer
(within Apache httpd
+ mod_wsgi)
SOAP Request
Transport: HTTP
ms_POST Request
Transport: HTTP via
a UNIX socket
Response
SOAP Reply

2.11.1 Creating Coverages

As we allready 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 132).

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 unambigous 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()
http_storage.save()

data_item.storage = http_storage
```

(continues on next page)

(continued from previous page)

```
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`.

2.11.2 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
```

2.11.3 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`[77] which can be chained to further refine the actual query. There is a lot of documentation on the topic[78] 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:

[77] https://docs.djangoproject.com/en/2.2/ref/models/querysets/#django.db.models.query.QuerySet
[78] https://docs.djangoproject.com/en/dev/topics/db/queries/

```
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))
)
```

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.

2.11.4 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[79]:

```
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`* (page 128). 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`* (page 128) implementing either the *`FileStorageInterface`* (page 112), *`ConnectedStorageInterface`* (page 112) or the *`PackageInterface`* (page 112). See the *documentation for writing Plugins* (page 82) 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.

[79] https://docs.djangoproject.com/en/dev/ref/models/querysets/#std:fieldlookup-startswith

```
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 112) it behaves like the `retrieve()` function.

2.12 Processes

This section deals with the creation of new Processes to be exposed via the WPS interface.

Processes are simply *Components* (page 128) that implement the *ProcessInterface* (page 180). For general information about Plugins/Components please refer to the *Plugins* (page 82) documentation.

2.12.1 Creating a new Process

As we already mentioned, Processes are *Components* (page 128):

```
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"
        )),
    ]
```

(continues on next page)

(continued from previous page)

```
    ...
```

`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`[80] of outputs where the `keys` are the names of our declared outputs:

```
class ExampleProcess(Component):
    implements(ProcessInterface)

    ...

    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]]),
```

(continues on next page)

[80] https://docs.python.org/3.6/library/stdtypes.html#dict

(continued from previous page)

```
        )),
    ]
    outputs = [
        ("example_bbox_output", BoundingBoxData(
            "example_bbox_output", crss=(4326, 0)
        )),
    ]
    ...
```

The third kind of input and output is *ComplexData* (page 172) which can come in various formats, binary or textual representation and either raw or base64 encoding.

2.13 Asynchronous Task Processing - Developers Guide

Table of Contents

2.13.1 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 “atp_sum”. The database model used in by the ATP subsystem

is described in "*Task Tracker Data Model* (page 81)". The complete API reference can be found in "`eoxserver.resources.processes.tracker`".

2.13.2 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" )
```

2.13.3 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 "atp_sum".

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.

2.14 Testing

TBD

`eoxserver.testing.core`

2.15 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.

2.15.1 Fundamentals

Above all rules, adhere the rules defined in the Python PEP 8[81]. 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!
```

2.15.2 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

class ClassB():
    pass
```

A developer using these functions can choose to use the namespace explicitly:

[81] https://www.python.org/dev/peps/pep-0008/

```
from somepackage import myNS

myNS.functionA()
c = myNS.ClassB()
```

2.15.3 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 198) package root.

Use single empty lines to separate these import groups.

2.15.4 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 comlications 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.

CHAPTER 3

Release Notes

Release notes from various versions of EOxServer.

3.1 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.

3.2 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

3.3 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.

3.3.1 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[82] 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 106).

3.3.2 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[83] 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.

3.3.3 Miscallaneus 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[84] was implemented which is an order of magnitude faster than the previous dom[85] based solution.

3.3.4 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.

[82] https://docs.djangoproject.com/en/dev/ref/models/querysets/
[83] http://trac.edgewall.org/wiki/TracDev/ComponentArchitecture
[84] http://lxml.de/
[85] https://docs.python.org/2/library/xml.dom.html

3.3.5 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.

WCS 2.0

EOxServer now fully supports the following WCS 2.0 service extensions:

- Scaling Extension[86]
- Interpolation Extension[87]
- RangeSubsetting Extension[88]
- CRS Extension[89]
- GeoTIFF Encoding Extension[90]

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.

3.3.6 Webclient

The existing webclient was replaced by a custom build of EOxClient[91]. It allows the inspection of more than one collection or dataset and features a dynamic timeline to ease the visual inspection of large datasets.

[86] https://portal.opengeospatial.org/files/12-039
[87] https://portal.opengeospatial.org/files/12-049
[88] https://portal.opengeospatial.org/files/12-040
[89] https://portal.opengeospatial.org/files/11-053
[90] https://portal.opengeospatial.org/files/?artifact_id=54813
[91] https://github.com/EOX-A/EOxClient

CHAPTER 4

API Reference

4.1 Subpackages

4.1.1 eoxserver.backends package

Subpackages

eoxserver.backends.packages package

Submodules

eoxserver.backends.packages.tar module

eoxserver.backends.packages.zip module

Module contents

eoxserver.backends.storages package

Submodules

eoxserver.backends.storages.ftp module

eoxserver.backends.storages.http module

eoxserver.backends.storages.local module

eoxserver.backends.storages.rasdaman module

Module contents

Submodules

eoxserver.backends.access module

eoxserver.backends.cache module

class eoxserver.backends.cache.**CacheContext**(*retention_time=None*, *cache_directory=None*, *managed=False*)

Bases: object[92]

Context manager to manage cached files.

add_mapping(*path*, *item*)
Add an external file to this context. Those files will be treated as if they are "within" the caches directory, but will not be clead up afterwards.

add_path(*cache_path*)
Add a path to this cache context. Also creates necessary sub-directories.

cache_directory
Returns the configured cache directory.

cleanup()
Perform cache cleanup.

contains(*cache_path*)
Check whether or not the path is contained in this cache.

relative_path(*cache_path*)
Returns a path relative to the cache directory.

exception eoxserver.backends.cache.**CacheException**
Bases: Exception[93]

eoxserver.backends.cache.**get_cache_context**()
Get the thread local cache context for this session. Raises an exception if the session was not initialized.

eoxserver.backends.cache.**set_cache_context**(*cache_context*)
Sets the cache context for this session. Raises an exception if there was already a cache context associated.

eoxserver.backends.cache.**setup_cache_session**(*config=None*)
Initialize the cache context for this session. If a cache context was already present, an exception is raised.

eoxserver.backends.cache.**shutdown_cache_session**()
Shutdown the cache context for this session and trigger any pending cleanup actions required.

eoxserver.backends.component module

class eoxserver.backends.component.**BackendComponent**(**args*)
Bases: *eoxserver.core.component.Component* (page 128)

This *Component* (page 128) provides extension points and helpers to easily retrive Package and Storage components by their type names.

[92] https://docs.python.org/3.6/library/functions.html#object
[93] https://docs.python.org/3.6/library/exceptions.html#Exception

connected_storages
List of components that implement *eoxserver.backends.interfaces.ConnectedStorageInterface* (page 112)

file_storages
List of components that implement *eoxserver.backends.interfaces.FileStorageInterface* (page 112)

get_connected_storage_component(*storage_type*)
Retrieve a component implementing the *eoxserver.backends.interfaces.ConnectedStorageInterface* (page 112) with the desired `storage_type`.

Parameters **storage_type** – the desired storage type

Returns the desired storage component or `None`

get_file_storage_component(*storage_type*)
Retrieve a component implementing the *eoxserver.backends.interfaces.FileStorageInterface* (page 112) with the desired `storage_type`.

Parameters **storage_type** – the desired storage type

Returns the desired storage component or `None`

get_package_component(*format*)
Retrieve a component implementing the *eoxserver.backends.interfaces.PackageInterface* (page 112) with the desired `format`.

Parameters **format** – the desired package format

Returns the desired package component or `None`

get_storage_component(*storage_type*)
Retrieve a component implementing the *eoxserver.backends.interfaces.FileStorageInterface* (page 112) or *eoxserver.backends.interfaces.ConnectedStorageInterface* (page 112) with the desired `storage_type`.

Parameters **storage_type** – the desired storage type

Returns the desired storage component or `None`

packages
List of components that implement *eoxserver.backends.interfaces.PackageInterface* (page 112)

storages
Helper to retrieve components for all storage interfaces.

eoxserver.backends.config module

class eoxserver.backends.config.**CacheConfigReader**(*config*)
Bases: *eoxserver.core.decoders.config.Reader* (page 117)

directory

retention_time

section = 'backends'

eoxserver.backends.interfaces module

class eoxserver.backends.interfaces.**AbstractStorageInterface**
Bases: object[94]

name
Name of the storage implementation.

validate(*url*)
Validates the given storage locator and raises a django.core.exceptions.ValidationError[95] if errors occurred.

class eoxserver.backends.interfaces.**ConnectedStorageInterface**
Bases: *eoxserver.backends.interfaces.AbstractStorageInterface* (page 112)

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 112)

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
- **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 eoxserver.backends.interfaces.**PackageInterface**
Bases: object[96]

[94] https://docs.python.org/3.6/library/functions.html#object
[95] https://docs.djangoproject.com/en/2.2/ref/exceptions/#django.core.exceptions.ValidationError
[96] https://docs.python.org/3.6/library/functions.html#object

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.

eoxserver.backends.middleware module

class eoxserver.backends.middleware.**BackendsCacheMiddleware**
Bases: `object`[97]

A Django Request Middleware[98] to manage cache setup and teardown when a request is beeing processed.

process_exception(*request*, *exception*)

process_request(*request*)

process_response(*request*, *response*)

process_template_response(*request*, *response*)

eoxserver.backends.models module

eoxserver.backends.testbase module

eoxserver.backends.testbase.**withFTPServer**(*port=2121*, *directory=None*)

Module contents

4.1.2 eoxserver.contrib package

Submodules

[97] https://docs.python.org/3.6/library/functions.html#object
[98] https://docs.djangoproject.com/en/dev/topics/http/middleware/

eoxserver.contrib.gdal module

This module imports and initializes GDAL; i.e enables exceptions and registers all available drivers.

eoxserver.contrib.gdal.**config_env**(*env*, *fail_on_override=False*, *reset_old=True*)

eoxserver.contrib.gdal.**get_extent**(*ds*)
Gets the extent of the GDAL Dataset in the form (min-x, min-y, max-x, max-y).

eoxserver.contrib.gdal.**open_with_env**(*path*, *env*, *shared=True*)

eoxserver.contrib.gdal.**set_env**(*env*, *fail_on_override=False*, *return_old=False*)

eoxserver.contrib.gdal_array module

eoxserver.contrib.mapserver module

class eoxserver.contrib.mapserver.**Class**(*name*, *mapobj=None*)
Bases: *eoxserver.contrib.mapserver.classObj* (page 114)

class eoxserver.contrib.mapserver.**Layer**(*name*, *metadata=None*, *type=0*, *mapobj=None*)
Bases: *eoxserver.contrib.mapserver.MetadataMixIn* (page 114), *eoxserver.contrib.mapserver.layerObj* (page 115)

class eoxserver.contrib.mapserver.**Map**(*metadata=None*)
Bases: *eoxserver.contrib.mapserver.MetadataMixIn* (page 114), *eoxserver.contrib.mapserver.mapObj* (page 115)

dispatch(*request*)

exception eoxserver.contrib.mapserver.**MapServerException**(*message*, *locator*, *code=None*)
Bases: Exception[99]

class eoxserver.contrib.mapserver.**MetadataMixIn**(*metadata=None*)
Bases: object[100]

Mix-In for classes that wrap mapscript objects with associated metadata.

setMetaData(*key_or_params*, *value=None*, *namespace=None*)
Convenvience method to allow setting multiple metadata values with one call and optionally setting a 'namespace' for each entry.

class eoxserver.contrib.mapserver.**Style**(*name*, *mapobj=None*)
Bases: *eoxserver.contrib.mapserver.styleObj* (page 115)

class eoxserver.contrib.mapserver.**classObj**
Bases: object[101]

class eoxserver.contrib.mapserver.**colorObj**
Bases: object[102]

eoxserver.contrib.mapserver.**config_env**(*map_obj*, *env*, *fail_on_override=False*, *reset_old=True*)

eoxserver.contrib.mapserver.**create_request**(*values*, *request_type=0*)
Creates a mapserver request from

[99] https://docs.python.org/3.6/library/exceptions.html#Exception
[100] https://docs.python.org/3.6/library/functions.html#object
[101] https://docs.python.org/3.6/library/functions.html#object
[102] https://docs.python.org/3.6/library/functions.html#object

eoxserver.contrib.mapserver.**dispatch**(*map_*, *request*)
Wraps the OWSDispatch method. Perfoms all necessary steps for a further handling of the result.

eoxserver.contrib.mapserver.**gdalconst_to_imagemode**(*const*)
This function translates a GDAL data type constant as defined in the gdalconst module to a MapScript image mode constant.

eoxserver.contrib.mapserver.**gdalconst_to_imagemode_string**(*const*)
This function translates a GDAL data type constant as defined in the gdalconst module to a string as used in the MapServer map file to denote an image mode.

class eoxserver.contrib.mapserver.**layerObj**
Bases: object[103]

class eoxserver.contrib.mapserver.**mapObj**
Bases: object[104]

eoxserver.contrib.mapserver.**setMetaData**(*obj*, *key_or_params*, *value=None*, *namespace=None*)
Convenvience function to allow setting multiple metadata values with one call and optionally setting a 'namespace' for each entry.

eoxserver.contrib.mapserver.**set_env**(*map_obj*, *env*, *fail_on_override=False*, *return_old=False*)

eoxserver.contrib.mapserver.**set_metadata**(*obj*, *key_or_params*, *value=None*, *namespace=None*)
Convenvience function to allow setting multiple metadata values with one call and optionally setting a 'namespace' for each entry.

class eoxserver.contrib.mapserver.**shapeObj**
Bases: object[105]

class eoxserver.contrib.mapserver.**styleObj**
Bases: object[106]

eoxserver.contrib.ogr module

eoxserver.contrib.osr module

class eoxserver.contrib.osr.**SpatialReference**(*raw=None*, *format=None*)
Bases: object[107]

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

[103] https://docs.python.org/3.6/library/functions.html#object
[104] https://docs.python.org/3.6/library/functions.html#object
[105] https://docs.python.org/3.6/library/functions.html#object
[106] https://docs.python.org/3.6/library/functions.html#object
[107] https://docs.python.org/3.6/library/functions.html#object

xml

eoxserver.contrib.vrt module

eoxserver.contrib.vsi module

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.

4.1.3 eoxserver.core package

Subpackages

eoxserver.core.decoders package

Submodules

eoxserver.core.decoders.base module

This module provides base functionality for any other decoder class.

class eoxserver.core.decoders.base.**BaseParameter**(*type=None*, *num=1*, *default=None*)
Bases: property[108]

Abstract base class for XML, KVP or any other kind of parameter.

fget(*decoder*)
Property getter function.

locator

select(*decoder*)
Interface method.

eoxserver.core.decoders.config module

This module contains facilities to help decoding configuration files. It relies on the ConfigParser module for actually reading the file.

class eoxserver.core.decoders.config.**Option**(*key=None*, *type=None*, *separator=None*, *required=False*, *default=None*, *section=None*, *doc=None*)
Bases: property[109]

The *Option* (page 116) is used as a property[110] for *Reader* (page 117) subclasses.

Parameters

- **key** – the lookup key; defaults to the property name of the *Reader* (page 117).

[108] https://docs.python.org/3.6/library/functions.html#property
[109] https://docs.python.org/3.6/library/functions.html#property
[110] https://docs.python.org/3.6/library/functions.html#property

- **type** – the type to parse the raw value; by default the raw string is returned
- **separator** – the separator for list options; by default no list is assumed
- **required** – if `True` raise an error if the option does not exist
- **default** – the default value
- **section** – override the section for this option

check(*reader*)

fget(*reader*)

class eoxserver.core.decoders.config.**Reader**(*config*)

Bases: `object`[111]

Base class for config readers.

Parameters **config** – an instance of `ConfigParser.RawConfigParser`

Readers should be used as such:

```
from ConfigParser import RawConfigParser
try:
    from cStringIO import StringIO
except ImportError:
    from io import StringIO
from textwrap import dedent
from eoxserver.core.decoders import config

class ExampleReader(config.Reader):
    section = "example_section"
    string_opt = config.Option()
    string_list_opt = config.Option(separator=",")
    integer_opt = config.Option(type=int)

    section = "other_section"
    mandatory_opt = config.Option(required=True)
    optional_opt = config.Option(default="some_default")

    special_section_opt = config.Option(section="special_section")

f = StringIO(dedent('''
    [example_section]
    string_opt = mystring
    string_list_opt = my,string,list
    integer_opt = 123456
    [other_section]
    mandatory_opt = mandatory_value
    # optional_opt = no value

    [special_section]
    special_section_opt = special_value
'''))

parser = RawConfigParser()
parser.readfp(f)
reader = ExampleReader(parser)
```

(continues on next page)

[111] https://docs.python.org/3.6/library/functions.html#object

(continued from previous page)

```
print reader.string_opt
print reader.string_list_opt
print reader.integer_opt
print reader.mandatory_opt
print reader.optional_opt
...
```

section = None

class eoxserver.core.decoders.config.**ReaderMetaclass**(*name*, *bases*, *dct*)
Bases: type[112]

eoxserver.core.decoders.config.**section**(*name*)
Helper to set the section of a *Reader* (page 117).

eoxserver.core.decoders.kvp module

This module contains facilities to help decoding KVP strings.

class eoxserver.core.decoders.kvp.**Decoder**(*params*)
Bases: object[113]

Base class for KVP decoders.

Parameters params – an instance of either dict[114], django.http.QueryDict[115] or basestring (which will be parsed using cgi.parse_qs()[116])

Decoders should be used as such:

```
from eoxserver.core.decoders import kvp
from eoxserver.core.decoders import typelist

class ExampleDecoder(kvp.Decoder):
    mandatory_param = kvp.Parameter(num=1)
    list_param = kvp.Parameter(type=typelist(separator=","))
    multiple_param = kvp.Parameter("multi", num="+")
    optional_param = kvp.Parameter(num="?", default="default_value")

decoder = ExampleDecoder(
    "mandatory_param=value"
    "&list_param=a,b,c"
    "&multi=a&multi=b&multi=c"
)

print decoder.mandatory_param
print decoder.list_param
print decoder.multiple_param
print decoder.optional_param
```

class eoxserver.core.decoders.kvp.**DecoderMetaclass**(*name*, *bases*, *dct*)
Bases: type[117]

[112] https://docs.python.org/3.6/library/functions.html#type
[113] https://docs.python.org/3.6/library/functions.html#object
[114] https://docs.python.org/3.6/library/stdtypes.html#dict
[115] https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.QueryDict
[116] https://docs.python.org/3.6/library/cgi.html#cgi.parse_qs
[117] https://docs.python.org/3.6/library/functions.html#type

Metaclass for KVP Decoders to allow easy parameter declaration.

class eoxserver.core.decoders.kvp.**MultiParameter**(*selector*, *num=1*, *default=None*, *locator=None*)

Bases: *eoxserver.core.decoders.kvp.Parameter* (page 119)

Class for selecting different KVP parameters at once.

Parameters

- **selector** – a function to determine if a key is used for the multi parameter selection
- **num** – defines how many times the key can be present; use any numeric value to set it to a fixed count, "*" for any number, "?" for zero or one time or "+" for one or more times
- **default** – the default value
- **locator** – override the locator in case of exceptions

select(*decoder*)

Interface method.

class eoxserver.core.decoders.kvp.**Parameter**(*key=None*, *type=None*, *num=1*, *default=None*, *locator=None*)

Bases: *eoxserver.core.decoders.base.BaseParameter* (page 116)

Parameter for KVP values.

Parameters

- **key** – the lookup key; defaults to the property name of the *Decoder* (page 118)
- **type** – the type to parse the raw value; by default the raw string is returned
- **num** – defines how many times the key can be present; use any numeric value to set it to a fixed count, "*" for any number, "?" for zero or one time or "+" for one or more times
- **default** – the default value
- **locator** – override the locator in case of exceptions

key = None

locator

select(*decoder*, *decoder_class=None*)

Interface method.

eoxserver.core.decoders.xml module

This module contains facilities to help decoding XML structures.

class eoxserver.core.decoders.xml.**Decoder**(*tree*)

Bases: `object`[118]

Base class for XML Decoders.

param params an instance of either `lxml.etree.ElementTree`, or `basestring` (which will be parsed using `lxml.etree.fromstring()`)

Decoders should be used as such:

[118] https://docs.python.org/3.6/library/functions.html#object

```
from eoxserver.core.decoders import xml
from eoxserver.core.decoders import typelist

class ExampleDecoder(xml.Decoder):
    namespaces = {"myns": "http://myns.org"}
    single = xml.Parameter("myns:single/text()", num=1)
    items = xml.Parameter("myns:collection/myns:item/text()", num="+")
    attr_a = xml.Parameter("myns:object/@attrA", num="?")
    attr_b = xml.Parameter("myns:object/@attrB", num="?", default="x")


decoder = ExampleDecoder('''
    <myns:root xmlns:myns="http://myns.org">
        <myns:single>value</myns:single>
        <myns:collection>
            <myns:item>a</myns:item>
            <myns:item>b</myns:item>
            <myns:item>c</myns:item>
        </myns:collection>
        <myns:object attrA="value"/>
    </myns:root>
''')

print decoder.single
print decoder.items
print decoder.attr_a
print decoder.attr_b
```

namespaces = {}

class eoxserver.core.decoders.xml.**Parameter**(*selector*, *type=None*, *num=1*, *default=None*, *namespaces=None*, *locator=None*)

Bases: *eoxserver.core.decoders.base.BaseParameter* (page 116)

Parameter for XML values.

Parameters

- **selector** – the node selector; if a string is passed it is interpreted as an XPath expression, a callable will be called with the root of the element tree and shall yield any number of node
- **type** – the type to parse the raw value; by default the raw string is returned
- **num** – defines how many times the key can be present; use any numeric value to set it to a fixed count, "*" for any number, "?" for zero or one time or "+" for one or more times
- **default** – the default value
- **namespaces** – any namespace necessary for the XPath expression; defaults to the *Decoder* (page 119) namespaces.
- **locator** – override the locator in case of exceptions

locator

select(*decoder*)

Interface method.

Module contents

class eoxserver.core.decoders.**Choice**(**choices*)
Bases: object[119]

Tries all given choices until one does return something.

class eoxserver.core.decoders.**Concatenate**(**choices*, ***kwargs*)
Bases: object[120]

Helper to concatenate the results of all sub-parameters to one.

exception eoxserver.core.decoders.**DecodingException**(*message*, *locator=None*)
Bases: Exception[121]

Base Exception class to be thrown whenever a decoding failed.

class eoxserver.core.decoders.**Exclusive**(**choices*)
Bases: object[122]

For mutual exclusive Parameters.

exception eoxserver.core.decoders.**ExclusiveException**(*message*, *locator=None*)
Bases: *eoxserver.core.decoders.DecodingException* (page 121)

exception eoxserver.core.decoders.**InvalidParameterException**(*message*, *locator=None*)
Bases: *eoxserver.core.decoders.DecodingException* (page 121)

code = 'InvalidParameterValue'

exception eoxserver.core.decoders.**MissingParameterException**(*locator*)
Bases: *eoxserver.core.decoders.DecodingException* (page 121)

Exception to be thrown, when a decoder could not read one parameter, where exactly one was required.

code = 'MissingParameterValue'

exception eoxserver.core.decoders.**MissingParameterMultipleException**(*locator*)
Bases: *eoxserver.core.decoders.DecodingException* (page 121)

Exception to be thrown, when a decoder could not read at least one parameter, where one ore more were required.

code = 'MissingParameterValue'

exception eoxserver.core.decoders.**NoChoiceResultException**(*message*, *locator=None*)
Bases: *eoxserver.core.decoders.DecodingException* (page 121)

exception eoxserver.core.decoders.**WrongMultiplicityException**(*locator*, *expected*, *result*)
Bases: *eoxserver.core.decoders.DecodingException* (page 121)

Decoding Exception to be thrown when the multiplicity of a parameter did not match the expected count.

code = 'InvalidParameterValue'

eoxserver.core.decoders.**boolean**(*raw*)
Functor to convert "true"/"false" to a boolean.

[119] https://docs.python.org/3.6/library/functions.html#object
[120] https://docs.python.org/3.6/library/functions.html#object
[121] https://docs.python.org/3.6/library/exceptions.html#Exception
[122] https://docs.python.org/3.6/library/functions.html#object

class eoxserver.core.decoders.**enum**(*values*, *case_sensitive=True*, *error_class=<class 'ValueError'>*)
Bases: object[123]

Helper for parameters that are expected to be in a certain enumeration. A ValueError is raised if not.

class eoxserver.core.decoders.**fixed**(*value*, *case_sensitive=True*)
Bases: object[124]

Helper for parameters that are expected to be have a fixed value and raises a ValueError if not.

eoxserver.core.decoders.**lower**(*value*)
Functor to return a lower-case string.

eoxserver.core.decoders.**strip**(*value*)
Functor to return a whitespace stripped string.

eoxserver.core.decoders.**to_dict**(*decoder*, *dict_class=<class 'dict'>*)
Utility function to get a dictionary representation of the given decoder. This function invokes all decoder parameters and sets the dictionary fields accordingly

class eoxserver.core.decoders.**typelist**(*typ=None*, *separator=' '*)
Bases: object[125]

Helper for XMLDecoder schemas that expect a string that represents a list of a type separated by some separator.

eoxserver.core.decoders.**upper**(*value*)
Functor to return a upper-case string.

class eoxserver.core.decoders.**value_range**(*min*, *max*, *type=<class 'float'>*)
Bases: object[126]

Helper to assert that a given parameter is within a specified range.

eoxserver.core.util package

Submodules

eoxserver.core.util.functools module

`total_ordering` and `force_total_ordering` are class decorators for Python 2.6 & Python 3.

They provides *all* the rich comparison methods on a class by defining *any* one of '__lt__', '__gt__', '__le__', '__ge__'.

`total_ordering` fills in all unimplemented rich comparison methods, assuming at least one is implemented. `__lt__` is taken as the base comparison method on which the others are built, but if that is not available it will be constructed from the first one found.

`force_total_ordering` does the same, but having taken a comparison method as the base it fills in *all* the others - this overwrites additional comparison methods that may be implemented, guaranteeing consistent comparison semantics.

```
from total_ordering import total_ordering

@total_ordering
```

(continues on next page)

[123] https://docs.python.org/3.6/library/functions.html#object
[124] https://docs.python.org/3.6/library/functions.html#object
[125] https://docs.python.org/3.6/library/functions.html#object
[126] https://docs.python.org/3.6/library/functions.html#object

(continued from previous page)

```
class Something(object):
    def __init__(self, value):
        self.value = value
    def __lt__(self, other):
        return self.value < other.value
```

It also works with Python 2.5, but you need to do the wrapping yourself:

```
from total_ordering import total_ordering

class Something(object):
    def __init__(self, value):
        self.value = value
    def __lt__(self, other):
        return self.value < other.value

total_ordering(Something)
```

It would be easy to modify for it to work as a class decorator for Python 3.X and a metaclass for Python 2.X.

eoxserver.core.util.functools.**force_total_ordering**(*cls*)

eoxserver.core.util.functools.**total_ordering**(*cls*)

eoxserver.core.util.geotools module

eoxserver.core.util.importtools module

This module contains utilities to easily import hierarchies of packages and modules.

eoxserver.core.util.importtools.**easy_import**(*module_path*)
Utility function to import one or more modules via a given module path. The last component of the module path can also be a '*' or a '**' character string which imports all submodules of the package either recursively (with '**') or not (with '*').

Parameters **module_path** – a typical python module path in the dotted notation. wildcards can be appeded at the last level.

eoxserver.core.util.importtools.**import_modules**(*base_module_path*)
Helper function to import all direct submodules within a package. This function is not recursive.

Parameters **base_module_path** – the base module path in the dotted notation.

eoxserver.core.util.importtools.**import_recursive**(*base_module_path*)
Helper function to recursively import all submodules and packages.

Parameters **base_module_path** – the base module path in the dotted notation.

eoxserver.core.util.importtools.**import_string**(*dotted_path*)
Import a dotted module path and return the attribute/class designated by the last name in the path. Raise ImportError if the import failed.

eoxserver.core.util.iteratortools module

This module is an extension of the `iteratortools` module and provides additional helpers.

eoxserver.core.util.iteratortools.**pairwise**(*iterable*)
s -> (s0,s1), (s2,s3), (s4, s5), ...

eoxserver.core.util.iteratortools.**pairwise_iterative**(*iterable*)
s -> (s0,s1), (s1,s2), (s2, s3), ...

eoxserver.core.util.multiparttools module

This module contains implementation of MIME multipart packing and unpacking utilities.

The main benefit of the utilities over other methods of mutipart handling is that the functions of this module do not manipulate the input data buffers and especially avoid any unnecessary data copying.

eoxserver.core.util.multiparttools.**capitalize**(*header_name*)
Capitalize header field name. Eg., 'content-type' is capilalized to 'Content-Type'.

Deprecated since version 0.4.

eoxserver.core.util.multiparttools.**capitalize_header**(*key*)
Returns a capitalized version of the header line such as 'content-type' -> 'Content-Type'.

eoxserver.core.util.multiparttools.**getMimeType**(*content_type*)
Extract MIME-type from Content-Type string and convert it to lower-case.

Deprecated since version 0.4.

eoxserver.core.util.multiparttools.**getMultipartBoundary**(*content_type*)
Extract boundary string from mutipart Content-Type string.

Deprecated since version 0.4.

eoxserver.core.util.multiparttools.**get_substring**(*data*, *boundary*, *offset*, *end*)
Retrieves the substring of `data` until the next `boundary` from a given offset to a until `end`.

eoxserver.core.util.multiparttools.**iterate**(*data*, *offset=0*, *end=None*, *headers=None*)
Efficient generator function to iterate over a single- or multipart message. I yields tuples in the shape (`headers`, `data`), where headers is a `dict` and data a buffer object, referencing the subset of the original content. In case of multipart messages, the multipart headers are yielded beforehand, with an empty string as data.

The *offset* parameter specifies the offset index to the start of the data. This is mostly used in the recursive call. The same applies to the *end* parameter.

The *headers* parameter specifies that the header section of the response was already read, and the headers are now entailed in the given dict. If this parameter is omitted, the headers are read from the stream.

eoxserver.core.util.multiparttools.**mpPack**(*parts*, *boundary*)
Low-level memory-friendly MIME multipart packing.

Note: The data payload is passed untouched and no transport encoding of the payload is performed.

Inputs:

- **parts - list of part-tuples, each tuple shall have two elements** the header list and (string) payload. The header itsels should be a sequence of key-value pairs (tuples).
- boundary - boundary string

Ouput:

- list of strings (which can be directly passsed as a Django response content)

Deprecated since version 0.4.

eoxserver.core.util.multiparttools.**mpUnpack**(*cbuffer*, *boundary*, *capitalize=False*)
Low-level memory-friendly MIME multipart unpacking.

Note: The payload of the multipart package data is neither modified nor copied. No decoding of the transport encoded payload is performed.

Note: The subroutine does not unpack any nested mutipart content.

Inputs:

- `cbuffer` - character buffer (string) containing the the header list and (string) payload. The header itsels should be a sequence of key-value pairs (tuples).
- `boundary` - boundary string
- `capitalize` - by default the header keys are converted to lower-case (e.g., 'content-type'). To capitalize the names (e.g., 'Content-Type') set this option to true.

Output:

- list of parts - each part is a tuple of the header dictionary, payload `cbuffer` offset and payload size.

Deprecated since version 0.4.

eoxserver.core.util.multiparttools.**parse_parametrized_option**(*string*)
Parses a parametrized options string like 'base;option=value;otheroption=othervalue'.

Returns the base string and a `dict`[127] with all parameters

eoxserver.core.util.perftools module

class eoxserver.core.util.perftools.**DurationMeasurement**(*name*, *logger*, *level*)
Bases: `object`[128]

duration

eoxserver.core.util.perftools.**log_duration**(*name*, *logger=None*, *level=10*)
Convenience function to log the duration of a specific event.

Parameters

- **name** – The name of the event.
- **logger** – The logger to use.
- **level** – The log level to log the final message to.

eoxserver.core.util.rect module

This module contains definition of the auxiliary 2D bounding box class.

class eoxserver.core.util.rect.**Rect**
Bases: `tuple`[129]

Named tuple to describe areas in a 2D array like in images. The tuple is always in the form (offset_x, offset_y, size_x, size_y).

Parameters

[127] https://docs.python.org/3.6/library/stdtypes.html#dict
[128] https://docs.python.org/3.6/library/functions.html#object
[129] https://docs.python.org/3.6/library/stdtypes.html#tuple

- **offset_x** – the x offset of the origin
- **offset_y** – the y offset of the origin
- **size_x** – thy x size of the rect
- **size_y** – thy y size of the rect
- **upper_x** – thy upper x offset of the rect (mutually exclusive with size_x)
- **upper_y** – thy upper y offset of the rect (mutually exclusive with size_y)

area

envelope(*other*)
Returns the envelope of two *Rect* (page 125), i.e., a smallest rectange contaning the input rectangles.

intersection(*other*)
Returns the intersection of two *Rect* (page 125), i.e., a largest common rectanle contained by the input rectangles.

intersects(*other*)
Tests whether two *Rect* (page 125) overlap (True) or not (False).

offset

offset_x

offset_y

size

size_x

size_y

translated(*tup*)
Returns a new *Rect* (page 125) shifted by the given offset.

upper

upper_x

upper_y

eoxserver.core.util.timetools module

eoxserver.core.util.timetools.**isoformat**(*dt*)
Formats a datetime object to an ISO string. Timezone naive datetimes are are treated as UTC Zulu. UTC Zulu is expressed with the proper "Z" ending and not with the "+00:00" offset declaration.

Parameters **dt** – the datetime.datetime[130] to encode

Returns an encoded string

eoxserver.core.util.timetools.**parse_duration**(*value*)
Parses an ISO 8601 duration string into a python timedelta object. Raises a *ValueError* if a conversion was not possible.

eoxserver.core.util.timetools.**parse_iso8601**(*value*, *tzinfo=None*)
Parses an ISO 8601 date or datetime string to a python date or datetime. Raises a *ValueError* if a conversion was not possible. The returned datetime is always considered time-zone aware and defaulting to the given timezone *tzinfo* or UTC Zulu if none was specified.

[130] https://docs.python.org/3.6/library/datetime.html#datetime.datetime

If the optional module `dateutil` is installed, it is used in preference over the `dateparse`[131] functions.

Parameters

- **value** – the string value to be parsed
- **tzinfo** – an optional tzinfo object that is used when the input string is not timezone aware

Returns a `datetime.datetime`[132]

eoxserver.core.util.xmltools module

This module contains utils for XML encoding, decoding and printing.

class eoxserver.core.util.xmltools.**NameSpace**(*uri*, *prefix=None*, *schema_location=None*)
Bases: `object`[133]

Helper object to ease the dealing with namespaces in both encoding and decoding.

Parameters

- **uri** – the namespace URI
- **prefix** – the namespace prefix
- **schema_location** – the schema location of this namespace

prefix

schema_location

uri

class eoxserver.core.util.xmltools.**NameSpaceMap**(**namespaces*)
Bases: `dict`[134]

Helper object to ease the setup and management of namespace collections in both encoding and decoding. Can (and should) be passed as `namespaces` attribute in *eoxserver.core.decoders.xml.Decoder* (page 119) subclasses.

Parameters **namespaces** – a list of *NameSpace* (page 127) objects.

add(*namespace*)

schema_locations

class eoxserver.core.util.xmltools.**XMLEncoder**
Bases: `object`[135]

Base class for XML encoders using lxml.etree. This class does not actually provide any helpers for encoding XML in a tree structure (this is already done in lxml.etree), but adds tree to string serialization and automatic handling of schema locations.

content_type

get_schema_locations()
Interface method. Returns a dict mapping namespace URIs to a network locations.

serialize(*tree*, *pretty_print=True*, *encoding='iso-8859-1'*)
Serialize a tree to an XML string. Also adds the `schemaLocations` attribute to the root node.

[131] https://docs.djangoproject.com/en/2.2/ref/utils/#module-django.utils.dateparse
[132] https://docs.python.org/3.6/library/datetime.html#datetime.datetime
[133] https://docs.python.org/3.6/library/functions.html#object
[134] https://docs.python.org/3.6/library/stdtypes.html#dict
[135] https://docs.python.org/3.6/library/functions.html#object

eoxserver.core.util.xmltools.**add_cdata**(*element*, *cdata*)

eoxserver.core.util.xmltools.**parse**(*obj*)
Helper function to parse XML either directly from a string, or fall back to whatever `lxml.etree.parse` parses. Returns `None` if it could not parse any XML.

Module contents

Submodules

eoxserver.core.component module

class eoxserver.core.component.**Component**(**args*)
Bases: `object`[136]

Base class for components.

Every component can declare what extension points it provides, as well as what extension points of other components it extends.

static implements(**interfaces*)
Can be used in the class definition of *Component* subclasses to declare the extension points that are extended.

class eoxserver.core.component.**ExtensionPoint**(*interface*)
Bases: `property`[137]

Marker class for extension points in components.

extensions(*component*)
Return a list of components that declare to implement the extension point interface.

class eoxserver.core.component.**UniqueExtensionPoint**(*interface*)
Bases: *eoxserver.core.component.ExtensionPoint* (page 128)

Marker class for unique extension points in components.

extensions(*component*)
Return the single component that is implementing the interaface. If none is found, or more than one, an exception is raised.

eoxserver.core.component.**implements**(**interfaces*)
Can be used in the class definition of *Component* subclasses to declare the extension points that are extended.

class eoxserver.core.component.**Interface**
Bases: `object`[138]

Marker base class for extension point interfaces.

exception eoxserver.core.component.**ComponentException**
Bases: `Exception`[139]

class eoxserver.core.component.**ComponentManager**
Bases: `object`[140]

The component manager keeps a pool of active components.

[136] https://docs.python.org/3.6/library/functions.html#object
[137] https://docs.python.org/3.6/library/functions.html#property
[138] https://docs.python.org/3.6/library/functions.html#object
[139] https://docs.python.org/3.6/library/exceptions.html#Exception
[140] https://docs.python.org/3.6/library/functions.html#object

component_activated(*component*)
Can be overridden by sub-classes so that special initialization for components can be provided.

disable_component(*component*)
Force a component to be disabled.

Parameters **component** – can be a class or an instance.

is_component_enabled(*cls*)
Can be overridden by sub-classes to veto the activation of a component.

If this method returns *False*, the component was disabled explicitly. If it returns *None*, the component was neither enabled nor disabled explicitly. In both cases, the component with the given class will not be available.

is_enabled(*cls*)
Return whether the given component class is enabled.

eoxserver.core.config module

This module provides an implementation of a system configuration that relies on different configuration files.

eoxserver.core.config.**get_eoxserver_config**()
Returns the EOxServer config as a `ConfigParser.RawConfigParser`

eoxserver.core.config.**get_instance_config_path**()
Convenience function to get the path to the instance config.

eoxserver.core.config.**reload_eoxserver_config**()
Triggers the loading or reloading of the EOxServer config as a `ConfigParser.RawConfigParser`.

eoxserver.core.management module

exception eoxserver.core.management.**CommandNotFound**(*cmdname*)
Bases: `Exception`[141]

class eoxserver.core.management.**EOxServerAdminCommand**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)
Bases: `django.core.management.base.BaseCommand`

execute(**args*, ***kwargs*)
Try to execute this command, performing system checks if needed (as controlled by the `requires_system_checks` attribute, except if force-skipped).

eoxserver.core.management.**execute_from_commandline**()

eoxserver.core.management.**get_commands**()

eoxserver.core.management.**print_possible_commands**(*commands*, *stream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

[141] https://docs.python.org/3.6/library/exceptions.html#Exception

eoxserver.core.models module

eoxserver.core.views module

eoxserver.core.views.**index**(*request*)

Module contents

The eoxserver.core package provides functionality for the initialization and re-initialization of the component system. For convenience, the module imports the most important items from the *eoxserver.core.component* (page 128) module and instantiates a component manager `eoxserver.core.env`.

eoxserver.core.**initialize**()
Initialize the EOxServer plugin system by trying to import all the plugins referenced by the *PLUGINS* configuration entry from the settings module. If a module path ends with '*' then all direct submodules will be imported aswell and if it ends with '**' it means that the import will be done recursively.

eoxserver.core.**reset**()
Reset the EOxServer plugin system.

4.1.4 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

4.1.5 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[142]

[142] https://docs.python.org/3.6/library/functions.html#object

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 (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[143]

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[144]

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

This module provides CRS handling utilities.

[143] https://docs.python.org/3.6/library/functions.html#object
[144] https://docs.python.org/3.6/library/functions.html#object

class eoxserver.resources.coverages.crss.**CRSsConfigReader**(*config*)
Bases: *eoxserver.core.decoders.config.Reader* (page 117)

section = 'services.ows.wcs'

supported_crss_wcs

supported_crss_wms

eoxserver.resources.coverages.crss.**EPSG_AXES_REVERSED = {2036, 2044, 2045, 2065, 2081, 208**
Set (Python `set` type) of EPSG codes of CRS whose axes are displayed in reversed order. Source: GDAL 1.10.0, WKT/AXES definitions

eoxserver.resources.coverages.crss.**asInteger**(*epsg*)
convert EPSG code to integer

eoxserver.resources.coverages.crss.**asProj4Str**(*epsg*)
convert EPSG code to *proj4* `+init=epsg:<code>` notation

eoxserver.resources.coverages.crss.**asShortCode**(*epsg*)
convert EPSG code to short CRS `EPSG:<code>` notation

eoxserver.resources.coverages.crss.**asURL**(*epsg*)
convert EPSG code to OGC URL CRS `http://www.opengis.net/def/crs/EPSG/0/<code>` notation

eoxserver.resources.coverages.crss.**asURN**(*epsg*)
convert EPSG code to OGC URN CRS `urn:ogc:def:crs:epsg::<code>` notation

eoxserver.resources.coverages.crss.**crs_bounds**(*srid*)
Get the maximum bounds of the CRS.

eoxserver.resources.coverages.crss.**crs_tolerance**(*srid*)
Get the "tolerance" of the CRS

eoxserver.resources.coverages.crss.**fromInteger**(*string*)
parse EPSG code from simple integer string

eoxserver.resources.coverages.crss.**fromProj4Str**(*string*)
parse EPSG code from given string in OGC Proj4Str CRS `+init=epsg:<code>` notation

eoxserver.resources.coverages.crss.**fromShortCode**(*string*)
parse EPSG code from given string in short CRS `EPSG:<code>` notation

eoxserver.resources.coverages.crss.**fromURL**(*string*)
parse EPSG code from given string in OGC URL CRS `http://www.opengis.net/def/crs/EPSG/0/<code>` notation

eoxserver.resources.coverages.crss.**fromURN**(*string*)
parse EPSG code from given string in OGC URN CRS `urn:ogc:def:crs:epsg::<code>` notation

eoxserver.resources.coverages.crss.**getAxesSwapper**(*epsg*, *swapAxes=None*)
Second order function returning point tuple axes swaper f(x,y) -> (x,y) or f(x,y) -> (y,x). The axes order is determined by the provided EPSG code. (Or exlicitely by the swapAxes boolean flag.

eoxserver.resources.coverages.crss.**getSupportedCRS_WCS**(*format_function=<function asShortCode>*)
Get list of CRSes supported by WCS. The `format_function` is used to format individual list items.

eoxserver.resources.coverages.crss.**getSupportedCRS_WMS**(*format_function=<function asShortCode>*)
Get list of CRSes supported by WMS. The `format_function` is used to format individual list items.

eoxserver.resources.coverages.crss.**hasSwappedAxes**(*epsg*)
Decide whether the coordinate system given by the passed EPSG code is displayed with swapped axes (True) or not (False).

eoxserver.resources.coverages.crss.**isProjected**(*epsg*)
Is the coordinate system projected (True) or Geographic (False)?

eoxserver.resources.coverages.crss.**is_image_crs**(*string*)

eoxserver.resources.coverages.crss.**parseEPSGCode**(*string*, *parsers*)
parse EPSG code using provided sequence of EPSG parsers

eoxserver.resources.coverages.crss.**validateEPSGCode**(*string*)
Check whether the given string is a valid EPSG code (True) or not (False)

eoxserver.resources.coverages.dateline module

eoxserver.resources.coverages.formats module

This module contains format handling utilities.

class eoxserver.resources.coverages.formats.**Format**(*mime_type*, *driver*, *extension*, *is_writeable*)
Bases: object[145]

Format record class. The class is rather structure with read-only properties (below). The class implements __str__() and __eq__() methods.

defaultExt
default extension (including dot)

driver
library/driver identifier

isWriteable
boolean flag indicating that output can be produced

mimeType
MIME-type

wcs10name
get WCS 1.0 format name

class eoxserver.resources.coverages.formats.**FormatConfigReader**(*config*)
Bases: *eoxserver.core.decoders.config.Reader* (page 117)

default_native_format

section = 'services.ows.wcs20'

source_to_native_format_map

supported_formats_wcs

supported_formats_wms

class eoxserver.resources.coverages.formats.**FormatRegistry**(*config*)
Bases: object[146]

[145] https://docs.python.org/3.6/library/functions.html#object
[146] https://docs.python.org/3.6/library/functions.html#object

The *FormatRegistry* (page 134) class represents cofiguration of file supported formats and of the auxiliary methods. The formats' configuration relies on two configuration files:

- the default formats' configuration (`eoxserver/conf/default_formats.conf`)
- the optional instance configuration (`conf/format.conf` in the instance directory)

Configuration values are read from these files.

getDefaultNativeFormat()
Get default nativeFormat as defined in section 'services.ows.wcs20'.

getFormatByMIME(*mime_type*)
Get format record for the given MIME type. In case of no match None is returned.

getFormatsAll()
Get list of all registered formats

getFormatsByDriver(*driver_name*)
Get format records for the given GDAL driver name. In case of no match empty list is returned.

getFormatsByWCS10Name(*wcs10name*)
Get format records for the given GDAL driver name. In case of no match an empty list is returned.

getSupportedFormatsWCS()
Get list of formats to be announced as supported WCS formats.

The the listed formats must be: * defined in EOxServers configuration (section "services.ows.wcs", item "supported_formats") * defined in the formats' configuration ("default_formats.conf" or "formats.conf") * supported by the used GDAL installation

getSupportedFormatsWMS()
Get list of formats to be announced as supported WMS formats.

The the listed formats must be: * defined in EOxServers configuration (section "services.ows.wms", item "supported_formats") * defined in the formats' configuration ("default_formats.conf" or "formats.conf") * supported by the used GDAL installation

mapSourceToNativeWCS20(*format*)
Map source format to WCS 2.0 native format.

Both the input and output shall be instances of `Formats` class. The input format can be obtained, e.g., by the *getFormatByDriver* or *getFormatByMIME* method.

To force the default native format use None as the source format.

The format mapping follows these rules:

1. Mapping based on the explicite rules is applied if possible (defined in EOxServers configuration, section "services.ows.wcs20", item "source_to_native_format_map"). If there is no mapping available the source format is kept.
2. If the format resulting from step 1 is not a writable GDAL format or it is not among the supported WCS formats than it is replaced by the default native format (defined in EOxServers configuration, section "services.ows.wcs20", item "default_native_format"). In case of writable GDAL format, the result of step 1 is returned.

exception eoxserver.resources.coverages.formats.**FormatRegistryException**
Bases: `Exception`[147]

eoxserver.resources.coverages.formats.**getFormatRegistry**()
Get initialised instance of the FormatRegistry class. This is the preferable way to get the Format Registry.

[147] https://docs.python.org/3.6/library/exceptions.html#Exception

eoxserver.resources.coverages.formats.**valDriver**(*string*)
Driver identifier reg.ex. validator. If pattern not matched 'None' is returned otherwise the input is returned.

eoxserver.resources.coverages.formats.**valMimeType**(*string*)
MIME type reg.ex. validator. If pattern not matched 'None' is returned otherwise the input is returned.

eoxserver.resources.coverages.models module

eoxserver.resources.coverages.rangetype module

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

4.1.6 eoxserver.services package

Subpackages

eoxserver.services.auth package

Submodules

eoxserver.services.auth.base module

This module contains basic classes and functions for the security layer (which is integrated in the service layer for now).

class eoxserver.services.auth.base.**AuthConfigReader**(*config*)
Bases: *eoxserver.core.decoders.config.Reader* (page 117)

allowLocal

attribute_mapping

authz_service

pdp_type

section = 'services.auth.base'

serviceID

class eoxserver.services.auth.base.**BasePDP**
Bases: object[148]

This is the base class for PDP implementations. It provides a skeleton for authorization request handling.

authorize(*request*)
This method handles authorization requests according to the requirements given in the PolicyDecisionPointInterface declaration.

Internally, it invokes the _decide() method that implements the actual authorization decision logic.

class eoxserver.services.auth.base.**PDPComponent**(**args*)
Bases: *eoxserver.core.component.Component* (page 128)

get_pdp(*pdp_type*)

pdps
List of components that implement *eoxserver.services.auth.interfaces.PolicyDecisionPointInterface* (page 137)

eoxserver.services.auth.base.**getPDP**()

eoxserver.services.auth.charonpdp module

eoxserver.services.auth.dummypdp module

eoxserver.services.auth.exceptions module

exception eoxserver.services.auth.exceptions.**AuthorisationException**
Bases: Exception[149]

code = 'AccessForbidden'

eoxserver.services.auth.interfaces module

class eoxserver.services.auth.interfaces.**PolicyDecisionPointInterface**
Bases: object[150]

This is the interface for Policy Decision Point (PDP) implementations.

authorize(*request*)
This method takes an OWSRequest object as input and returns an AuthorizationResponse 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.

[148] https://docs.python.org/3.6/library/functions.html#object
[149] https://docs.python.org/3.6/library/exceptions.html#Exception
[150] https://docs.python.org/3.6/library/functions.html#object

pdp_type
The type name of this PDP.

eoxserver.services.auth.middleware module

class eoxserver.services.auth.middleware.**PDPMiddleware**
Bases: object[151]

Middleware to allow authorization agains a Policy Decision Point. This middleware will be used for *all* requests and *all* configured views. If you only want to provide PDP authorization for a single view, use the *pdp_protect*.

process_view(*request*, *view_func*, *view_args*, *view_kwargs*)

eoxserver.services.auth.middleware.**pdp_protect**(*view*)
Wrapper function for views that shall be protected by PDP authorization. This function can be used as a decorator of a view function, or as a modifier to be used in the url configuration file. e.g:

```
urlpatterns = patterns('',
    ...
    url(r'^ows', pdp_protect(ows)),
    ...
)
```

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

[151] https://docs.python.org/3.6/library/functions.html#object

eoxserver.services.gml.v32.encoders module

class eoxserver.services.gml.v32.encoders.**EOP20Encoder**
Bases: *eoxserver.services.gml.v32.encoders.GML32Encoder* (page 139)

encode_earth_observation(*identifier*, *begin_time*, *end_time*, *footprint*, *contributing_datasets=None*, *subset_polygon=None*)

encode_footprint(*footprint*, *eo_id*)

encode_metadata_property(*eo_id*, *contributing_datasets=None*)

class eoxserver.services.gml.v32.encoders.**GML32Encoder**
Bases: object[152]

encode_linear_ring(*ring*, *sr*)

encode_multi_surface(*geom*, *base_id*)

encode_polygon(*polygon*, *base_id*)

encode_time_instant(*time*, *identifier*)

encode_time_period(*begin_time*, *end_time*, *identifier*)

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.connectors.**get_connector_by_test**(*coverage*, *data_items*)

Get a coverage metadata format reader by testing.

[152] https://docs.python.org/3.6/library/functions.html#object

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[153]

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[154]

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.

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.

[153] https://docs.python.org/3.6/library/functions.html#object
[154] https://docs.python.org/3.6/library/functions.html#object

class eoxserver.services.mapserver.interfaces.**StyleApplicatorInterface**
Bases: object[155]

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

class eoxserver.services.ows.common.v11.encoders.**OWS11ExceptionXMLEncoder**
Bases: *eoxserver.core.util.xmltools.XMLEncoder* (page 127)

encode_exception(*message*, *version*, *code*, *locator=None*)

get_schema_locations()
Interface method. Returns a dict mapping namespace URIs to a network locations.

[155] https://docs.python.org/3.6/library/functions.html#object

Module contents

eoxserver.services.ows.common.v20 package

Submodules

eoxserver.services.ows.common.v20.encoders module

class eoxserver.services.ows.common.v20.encoders.**OWS20Encoder**
Bases: *eoxserver.core.util.xmltools.XMLEncoder* (page 127)

encode_operations_metadata(*request*, *service*, *versions*)

encode_reference(*node_name*, *href*, *reftype='simple'*)

encode_service_identification(*service*, *conf*, *profiles*)

encode_service_provider(*conf*)

get_conf()

get_http_service_url(*request*)

class eoxserver.services.ows.common.v20.encoders.**OWS20ExceptionXMLEncoder**
Bases: *eoxserver.core.util.xmltools.XMLEncoder* (page 127)

encode_exception(*message*, *version*, *code*, *locator=None*)

get_schema_locations()
Interface method. Returns a dict mapping namespace URIs to a network locations.

eoxserver.services.ows.common.v20.exceptionhandler module

class eoxserver.services.ows.common.v20.exceptionhandler.**OWS20ExceptionHandler**
Bases: object[156]

A Fallback exception handler. This class does on purpose not implement the ExceptionHandlerInterface and must be instantiated manually.

handle_exception(*request*, *exception*)

Module contents

Submodules

eoxserver.services.ows.common.config module

class eoxserver.services.ows.common.config.**CapabilitiesConfigReader**(*config*)
Bases: *eoxserver.core.decoders.config.Reader* (page 117)

abstract

access_constraints

administrative_area

[156] https://docs.python.org/3.6/library/functions.html#object

city

contact_instructions

country

delivery_point

electronic_mail_address

fees

hours_of_service

http_service_url

individual_name

keywords

name

onlineresource

phone_facsimile

phone_voice

position_name

postal_code

provider_name

provider_site

role

section = 'services.ows'

title

update_sequence

class eoxserver.services.ows.common.config.**WCSEOConfigReader**(*config*)

Bases: *eoxserver.core.decoders.config.Reader* (page 117)

paging_count_default

section = 'services.ows.wcs20'

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

class eoxserver.services.ows.wcs.v10.exceptionhandler.**WCS10ExceptionHandler**
Bases: object[157]

handle_exception(*request*, *exception*)

request = None

service = 'WCS'

versions = ('1.0.0',)

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

class eoxserver.services.ows.wcs.v11.exceptionhandler.**WCS11ExceptionHandler**
Bases: object[158]

handle_exception(*request*, *exception*)

request = None

service = 'WCS'

versions = ('1.1.0', '1.1.1', '1.1.2')

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

[157] https://docs.python.org/3.6/library/functions.html#object
[158] https://docs.python.org/3.6/library/functions.html#object

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.encodings.**get_encoding_extensions**()

eoxserver.services.ows.wcs.v20.packages package

Submodules

eoxserver.services.ows.wcs.v20.packages.tar module

class eoxserver.services.ows.wcs.v20.packages.tar.**TarPackageWriter**

Bases: object[159]

Package writer for compressed and uncompressed tar files.

add_to_package(*package*, *data*, *size*, *location*)

cleanup(*package*)

create_package(*filename*, *format*, *params*)

get_file_extension(*package*, *format*, *params*)

get_mime_type(*package*, *format*, *params*)

supports(*format*, *params*)

eoxserver.services.ows.wcs.v20.packages.zip module

class eoxserver.services.ows.wcs.v20.packages.zip.**ZipPackageWriter**

Bases: object[160]

add_to_package(*package*, *data*, *size*, *location*)

cleanup(*package*)

create_package(*filename*, *format*, *params*)

get_file_extension(*package*, *format*, *params*)

get_mime_type(*package*, *format*, *params*)

[159] https://docs.python.org/3.6/library/functions.html#object
[160] https://docs.python.org/3.6/library/functions.html#object

supports(*format*, *params*)

Module contents

Submodules

eoxserver.services.ows.wcs.v20.describecoverage module

eoxserver.services.ows.wcs.v20.describeeocoverageset module

eoxserver.services.ows.wcs.v20.encoders module

eoxserver.services.ows.wcs.v20.exceptionhandler module

class eoxserver.services.ows.wcs.v20.exceptionhandler.**WCS20ExceptionHandler**(**args*)
Bases: *eoxserver.core.component.Component* (page 128)

handle_exception(*request*, *exception*)

request = None

service = 'WCS'

versions = ('2.0.0', '2.0.1')

eoxserver.services.ows.wcs.v20.getcapabilities module

eoxserver.services.ows.wcs.v20.getcoverage module

eoxserver.services.ows.wcs.v20.geteocoverageset 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[161]

parse_encoding_params(*request*)
Return a dict, containing all additional encoding parameters from a given request.

[161] https://docs.python.org/3.6/library/functions.html#object

supports (*format*, *options*)
Return a boolen value, whether or not an encoding extension supports a given format.

class eoxserver.services.ows.wcs.interfaces.**PackageWriterInterface**
Bases: object[162]

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 boolen value, whether or not a writer supports a given format.

class eoxserver.services.ows.wcs.interfaces.**WCSCapabilitiesRendererInterface**
Bases: object[163]

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[164]

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[165]

Interface for coverage renderers.

render (*params*)
Render the coverage with the given parameters.

[162] https://docs.python.org/3.6/library/functions.html#object
[163] https://docs.python.org/3.6/library/functions.html#object
[164] https://docs.python.org/3.6/library/functions.html#object
[165] https://docs.python.org/3.6/library/functions.html#object

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 149), *eoxserver.services.parameters.VersionedParams* (page 194)

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 149), *eoxserver.services.parameters.VersionedParams* (page 194)

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 149), *eoxserver.services.parameters.CapabilitiesRenderParams* (page 194)

class eoxserver.services.ows.wcs.parameters.**WCSParamsMixIn**
Bases: object[166]

Module contents

eoxserver.services.ows.wms package

[166] https://docs.python.org/3.6/library/functions.html#object

Subpackages

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

class eoxserver.services.ows.wms.v13.exceptionhandler.**WMS13Decoder**(*params*)
Bases: *eoxserver.core.decoders.kvp.Decoder* (page 118)

bgcolor
Property getter function.

exceptions
Property getter function.

format
Property getter function.

height
Property getter function.

width
Property getter function.

class eoxserver.services.ows.wms.v13.exceptionhandler.**WMS13ExceptionHandler**(**args*)
Bases: *eoxserver.core.component.Component* (page 128)

get_encoder(*request*)

handle_exception(*request*, *exception*)

request = None

service = 'WMS'

versions = ('1.3.0', '1.3')

class eoxserver.services.ows.wms.v13.exceptionhandler.**WMS13ExceptionImageEncoder**(*width=None*, *height=None*, *format=None*, *bgcolor=None*, *blank=False*)

Bases: object[167]

content_type

encode_exception(*message*, *code*, *locator=None*)

serialize(*image*)

class eoxserver.services.ows.wms.v13.exceptionhandler.**WMS13ExceptionXMLEncoder**

Bases: *eoxserver.core.util.xmltools.XMLEncoder* (page 127)

content_type

encode_exception(*message*, *code*, *locator=None*)

get_schema_locations()

Interface method. Returns a dict mapping namespace URIs to a network locations.

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: Exception[168]

[167] https://docs.python.org/3.6/library/functions.html#object
[168] https://docs.python.org/3.6/library/exceptions.html#Exception

code = 'InvalidCRS'

exception eoxserver.services.ows.wms.exceptions.**InvalidFormat**(*value*)
Bases: Exception[169]

code = 'InvalidFormat'

locator = 'format'

exception eoxserver.services.ows.wms.exceptions.**LayerNotDefined**(*layer*)
Bases: Exception[170]

code = 'LayerNotDefined'

locator = 'layers'

eoxserver.services.ows.wms.interfaces module

class eoxserver.services.ows.wms.interfaces.**WMSCapabilitiesRendererInterface**
Bases: object[171]

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[172]

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[173]

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[174]

Interface for WMS compatible map renderers.

[169] https://docs.python.org/3.6/library/exceptions.html#Exception
[170] https://docs.python.org/3.6/library/exceptions.html#Exception
[171] https://docs.python.org/3.6/library/functions.html#object
[172] https://docs.python.org/3.6/library/functions.html#object
[173] https://docs.python.org/3.6/library/functions.html#object
[174] https://docs.python.org/3.6/library/functions.html#object

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.

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

class eoxserver.services.ows.wps.parameters.allowed_values.**AllowedAny**
Bases: *eoxserver.services.ows.wps.parameters.allowed_values.BaseAllowed* (page 155)

Allowed values class allowing any value.

check(*value*)
check validity

verify(*value*)
Verify the value.

class eoxserver.services.ows.wps.parameters.allowed_values.**AllowedByReference**(*url*)
Bases: *eoxserver.services.ows.wps.parameters.allowed_values.BaseAllowed* (page 155)

Allowed values class defined by a reference.

NOTE: As it is not how such a reference definition looks like this class has the same behaviour as the AllowedAny class.

check(*value*)
check validity

url
Get the URL of the reference.

verify(*value*)
Verify the value.

class eoxserver.services.ows.wps.parameters.allowed_values.**AllowedEnum**(*values*, *dtype=<class 'eoxserver.services.ows.wps.*
Bases: *eoxserver.services.ows.wps.parameters.allowed_values.BaseAllowed*

(page 155), *eoxserver.services.ows.wps.parameters.allowed_values.TypedMixIn* (page 155)

Allowed values class allowing values from an enumerated set.

check(*value*)
check validity

values
Get the allowed values.

verify(*value*)
Verify the value.

class eoxserver.services.ows.wps.parameters.allowed_values.**AllowedRange**(*minval*, *maxval*, *closure='closed'*, *spacing=None*, *spacing_rtol=1e-09*, *dtype=<class 'eoxserver.services.ows.wps*

Bases: *eoxserver.services.ows.wps.parameters.allowed_values.BaseAllowed* (page 155), *eoxserver.services.ows.wps.parameters.allowed_values.TypedMixIn* (page 155)

Allowed values class allowing values from a range.

Constructor parameters: minval range lower bound - set to None if unbound maxval range upper bound - set to None if unbound closure *'closed'|'open'|'open-closed'|'closed-open' spacing uniform spacing of discretely sampled ranges spacing_rtol relative tolerance of the spacing match

ALLOWED_CLOSURES = ['closed', 'open', 'open-closed', 'closed-open']

check(*value*)
check validity

closure
Get the range closure type.

maxval
Get the upper bound of the range.

minval
Get the lower bound of the range.

spacing
Get the range spacing.

verify(*value*)
Verify the value.

class eoxserver.services.ows.wps.parameters.allowed_values.**AllowedRangeCollection**(**objs*)

Bases: *eoxserver.services.ows.wps.parameters.allowed_values.BaseAllowed* (page 155), *eoxserver.services.ows.wps.parameters.allowed_values.TypedMixIn* (page 155)

Allowed value class allowing values from a collection of AllowedEnum and AllowedRange instances.

check(*value*)
check validity

enum
Get merged set of the enumerated allowed values.

ranges
Get list of the allowed values' ranges.

verify(*value*)
Verify the value.

class eoxserver.services.ows.wps.parameters.allowed_values.**BaseAllowed**
Bases: object[175]

Allowed values base class.

check(*value*)
check validity

verify(*value*)
Verify the value.

class eoxserver.services.ows.wps.parameters.allowed_values.**TypedMixIn**(*dtype*)
Bases: object[176]

Mix-in class adding date-type to an allowed value range.

dtype
Get data-type.

eoxserver.services.ows.wps.parameters.base module

class eoxserver.services.ows.wps.parameters.base.**BaseParamMetadata**(*identifier*, *title=None*, *abstract=None*)
Bases: object[177]

Common metadata base of all parameter classes.

Constructor parameters: identifier item identifier title item title (human-readable name) abstract item abstract (human-readable description)

class eoxserver.services.ows.wps.parameters.base.**ParamMetadata**(*identifier*, *title=None*, *abstract=None*, *uom=None*, *crs=None*, *mime_type=None*, *encoding=None*, *schema=None*)
Bases: *eoxserver.services.ows.wps.parameters.base.BaseParamMetadata* (page 155)

Common metadata of the execute request parameters.

[175] https://docs.python.org/3.6/library/functions.html#object
[176] https://docs.python.org/3.6/library/functions.html#object
[177] https://docs.python.org/3.6/library/functions.html#object

Constructor parameters: identifier item identifier title item title (human-readable name) abstract item abstract (human-readable description) uom item LiteralData UOM crs item BoundingBox CRS mime_type item ComplexData mime-type encoding item ComplexData encoding schema item ComplexData schema

class eoxserver.services.ows.wps.parameters.base.**Parameter**(*identifier=None*, *title=None*, *abstract=None*, *metadata=None*, *optional=False*, *resolve_input_references=True*)

Bases: *eoxserver.services.ows.wps.parameters.base.BaseParamMetadata* (page 155)

Base parameter class used by the process definition.

Constructor parameters: identifier identifier of the parameter. title optional human-readable name (defaults to identifier). abstract optional human-readable verbose description. metadata optional metadata (title/URL dictionary). optional optional boolean flag indicating whether the input

parameter is optional or not.

resolve_input_references Set this option to False not to resolve input references. By default the references are resolved (downloaded and parsed) transparently. If set to False the references must be handled by the process.

eoxserver.services.ows.wps.parameters.bboxdata module

class eoxserver.services.ows.wps.parameters.bboxdata.**BoundingBox**(*bbox*, *crs=None*)

Bases: `tuple`[178]

Bounding-box class.

Constructor parameters:

bbox N-dimensional bounding box definition:

((xmin,),(xmax,)) ((xmin,ymin),(xmax,ymax)) ((xmin,ymin,zmin),(xmax,ymax,zmax))

or instance of the `Rect` class.

crs optional CRS identifier (URI)

as_rect
Cast to a *Rect* object. (Available only for the 2D bounding-box).

crs
Get the bounding-box CRS.

dimension
Get the bounding-box dimension.

lower
Get the bounding-box lower coordinates.

upper
Get the bounding-box upper coordinates.

[178] https://docs.python.org/3.6/library/stdtypes.html#tuple

class eoxserver.services.ows.wps.parameters.bboxdata.**BoundingBoxData**(*identifier*, *crss=None*, *dimension=2*, *default=None*, **args*, ***kwargs*)

Bases: *eoxserver.services.ows.wps.parameters.base.Parameter* (page 156)

Bounding-box parameter class

Constructor parameters: identifier identifier of the parameter. title optional human-readable name (defaults to identifier). abstract optional human-readable verbose description. metadata optional metadata (title/URL dictionary). optional optional boolean flag indicating whether the input

parameter is optional or not.

default optional default input value. Presence of the default value sets the parameter optional.

crss list of accepted CRSs (Coordinate Reference Systems). The CRSs shall be given in form of the integer EPSG codes. Defaults to WGS84 (EPSG:4326).

dimension optional dimension of the bounding box coordinates. Defaults to 2.

resolve_input_references Set this option to False not to resolve input references. By default the references are resolved (downloaded and parsed) transparently. If set to False the references must be handled by the process.

default_crs
Get the bounding-box default CRS.

dtype
alias of *eoxserver.services.ows.wps.parameters.data_types.Double* (page 164)

dtype_crs
alias of *eoxserver.services.ows.wps.parameters.crs.CRSType* (page 162)

classmethod encode_crs(*crs*)
Encode the output bounding CRS.

encode_kvp(*bbox*)
Encode KVP bounding box.

encode_xml(*bbox*)
Encode XML bounding box.

parse(*raw_bbox*)
Parse the input CRS.

classmethod parse_crs(*raw_crs*)
Parse the input bounding CRS.

eoxserver.services.ows.wps.parameters.codecs module

class eoxserver.services.ows.wps.parameters.codecs.**Codec**
Bases: object[179]

Base complex data codec.

[179] https://docs.python.org/3.6/library/functions.html#object

static decode(*file_in*, ***opt*)
Encoding generator.

static encode(*file_in*, ***opt*)
Encoding generator.

encoding = None

class eoxserver.services.ows.wps.parameters.codecs.**CodecBase64**
Bases: *eoxserver.services.ows.wps.parameters.codecs.Codec* (page 157)

Base64 codec

static decode(*file_in*, *urlsafe=False*, ***opt*)
Decoding generator.

static encode(*file_in*, *urlsafe=False*, ***opt*)
Encoding generator.

encoding = 'base64'

class eoxserver.services.ows.wps.parameters.codecs.**CodecRaw**
Bases: *eoxserver.services.ows.wps.parameters.codecs.Codec* (page 157)

Data encoder class.

static decode(*file_in*, ***opt*)
Decoding generator.

static encode(*file_in*, ***opt*)
Encoding generator.

encoding = None

eoxserver.services.ows.wps.parameters.complexdata module

class eoxserver.services.ows.wps.parameters.complexdata.**CDAsciiTextBuffer**(*data=''*, **args*, ***kwargs*)
Bases: *eoxserver.services.ows.wps.parameters.complexdata.CDByteBuffer* (page 159)

Complex data text (ASCII) in-memory buffer (StringIO). To be used to hold generic ASCII text. The text payload is stored as a byte-stream and this class cannot hold characters outside of the 7-bit ASCII characters' range.

Constructor parameters: data optional initial payload ASCII string mime_type ComplexData mime-type encoding ComplexData encoding schema ComplexData XML schema (applicable XML only) format an alternative format object defining the ComplexData

mime_type, encoding, and XML schema

filename optional raw output file-name set in the Content-Disposition HTTP header.

headers additional raw output HTTP headers encoded as a list of <key>, <value> pairs (tuples).

text_encoding optional keyword parameter defining the input text encoding. By default ASCII is assumed.

read(*size=None*)
Read at most size bytes, returned as a bytes object.

If the size argument is negative, read until EOF is reached. Return an empty bytes object at EOF.

write(*data*)
Write bytes to file.

Return the number of bytes written.

class eoxserver.services.ows.wps.parameters.complexdata.**CDBase**(*mime_type=None*, *encoding=None*, *schema=None*, *format=None*, *filename=None*, *headers=None*, ***kwargs*)

Bases: object[180]

Base class of the complex data container.

Constructor parameters (all optional and all defaulting to None): mime_type ComplexData mime-type encoding ComplexData encoding schema ComplexData XML schema (applicable XML only) format an alternative format object defining the ComplexData

mime_type, encoding, and XML schema

filename optional raw output file-name set in the Content-Disposition HTTP header.

headers additional raw output HTTP headers encoded as a list of <key>, <value> pairs (tuples).

data
Get the payload data.

encoding

class eoxserver.services.ows.wps.parameters.complexdata.**CDByteBuffer**(*data=b''*, **args*, ***kwargs*)

Bases: _io.BytesIO, *eoxserver.services.ows.wps.parameters.complexdata.CDBase* (page 159)

Complex data binary in-memory buffer (StringIO). To be used to hold a generic binary (byte-stream) payload.

Constructor parameters: data optional initial payload byte string mime_type ComplexData mime-type encoding ComplexData encoding schema ComplexData XML schema (applicable XML only) format an alternative format object defining the ComplexData

mime_type, encoding, and XML schema

filename optional raw output file-name set in the Content-Disposition HTTP header.

headers additional raw output HTTP headers encoded as a list of <key>, <value> pairs (tuples).

data
Get the payload data.

write(*data*)
Write bytes to file.

Return the number of bytes written.

[180] https://docs.python.org/3.6/library/functions.html#object

class eoxserver.services.ows.wps.parameters.complexdata.**CDFile**(*name*, *mode='rb'*, *buffering=-1*, **args*, ***kwargs*)

Bases: *eoxserver.services.ows.wps.parameters.complexdata.CDFileWrapper* (page 160)

Complex data file. To be used to hold a generic (binary or text) byte-stream payload. NOTE: The file allows you to specify whether the file is

temporary (will be automatically removed - by default) or permanent (preserved after object destruction).

Constructor parameters: name mandatory file-name mode opening mode (passed to *open*, 'r' by default) buffering buffering mode (passed to *open*, -1 by default) mime_type ComplexData mime-type encoding ComplexData encoding schema ComplexData XML schema (applicable XML only) format an alternative format object defining the ComplexData

mime_type, encoding, and XML schema

filename optional raw output file-name set in the Content-Disposition HTTP header.

remove_file optional keyword argument defining whether the file should be removed or not. Set to True by default.

class eoxserver.services.ows.wps.parameters.complexdata.**CDFileWrapper**(*file_object*, **args*, ***kwargs*)

Bases: *eoxserver.services.ows.wps.parameters.complexdata.CDBase* (page 159)

Complex data file (or file-like) object wrapper.

Constructor parameters: file_object mandatory seekable Python file or file-like object. mime_type ComplexData mime-type encoding ComplexData encoding schema ComplexData XML schema (applicable XML only) format an alternative format object defining the ComplexData

mime_type, encoding, and XML schema

filename optional raw output file-name set in the Content-Disposition HTTP header.

text_encoding optional source text file encoding

data

Get the payload data.

class eoxserver.services.ows.wps.parameters.complexdata.**CDObject**(*data*, **args*, ***kwargs*)

Bases: *eoxserver.services.ows.wps.parameters.complexdata.CDBase* (page 159)

Complex data wrapper around an arbitrary python object. To be used to set custom format attributes for the XML and JSON payload. NOTE: CDObject is not used for the input JSON and XML.

Constructor parameters: data mandatory object holding the payload data mime_type ComplexData mime-type encoding ComplexData encoding schema ComplexData XML schema (applicable XML only) format an alternative format object defining the ComplexData

mime_type, encoding, and XML schema

filename optional raw output file-name set in the Content-Disposition HTTP header.

headers additional raw output HTTP headers encoded as a list of <key>, <value> pairs (tuples).

data
Get the payload data.

class eoxserver.services.ows.wps.parameters.complexdata.**CDPermanentFile**(**args*, ***kwargs*)
Bases: *eoxserver.services.ows.wps.parameters.complexdata.CDFile* (page 159)

Complex data permanent file. To be used to hold a generic (binary or text) byte-stream payload. NOTE: This class preserves the actual file.

Constructor parameters: name mandatory file-name mode opening mode (passed to *open*, 'r' by default) buffering buffering mode (passed to *open*, -1 by default) mime_type ComplexData mime-type encoding ComplexData encoding schema ComplexData XML schema (applicable XML only) format an alternative format object defining the ComplexData

mime_type, encoding, and XML schema

filename optional raw output file-name set in the Content-Disposition HTTP header.

class eoxserver.services.ows.wps.parameters.complexdata.**CDTextBuffer**(*data=''*, **args*, ***kwargs*)
Bases: _io.StringIO, *eoxserver.services.ows.wps.parameters.complexdata.CDBase* (page 159)

Complex data text (Unicode) in-memory buffer (StringIO). To be used to hold generic text. The text payload is stored as a Unicode-stream.

Constructor parameters: data optional initial payload Unicode string mime_type ComplexData mime-type encoding ComplexData encoding schema ComplexData XML schema (applicable XML only) format an alternative format object defining the ComplexData

mime_type, encoding, and XML schema

filename optional raw output file-name set in the Content-Disposition HTTP header.

headers additional raw output HTTP headers encoded as a list of <key>, <value> pairs (tuples).

text_encoding optional keyword parameter defining the input text encoding. By default UTF-8 is assumed.

data
Get the payload data.

read(*size=None*)
Read at most size characters, returned as a string.

If the argument is negative or omitted, read until EOF is reached. Return an empty string at EOF.

write(*data*)
Write string to file.

Returns the number of characters written, which is always equal to the length of the string.

class eoxserver.services.ows.wps.parameters.complexdata.**ComplexData**(*identifier*, *formats*, **args*, ***kwargs*)
Bases: *eoxserver.services.ows.wps.parameters.base.Parameter* (page 156)

Complex-data parameter class

Constructor parameters: identifier identifier of the parameter. title optional human-readable name (defaults to identifier). abstract optional human-readable verbose description. metadata optional metadata (title/URL dictionary). optional optional boolean flag indicating whether the input

parameter is optional or not.

formats List of supported formats. resolve_input_references Set this option to False not to resolve

input references. By default the references are resolved (downloaded and parsed) transparently. If set to False the references must be handled by the process.

default_format
Get default the default format.

encode_raw(*data*)
encode complex data for raw output

encode_xml(*data*)
encode complex data to be embedded to an XML document

get_format(*mime_type*, *encoding=None*, *schema=None*)
Get format definition for the given mime-type and the optional encoding and schema.

parse(*data*, *mime_type*, *schema*, *encoding*, ***opt*)
parse input complex data

eoxserver.services.ows.wps.parameters.crs module

class eoxserver.services.ows.wps.parameters.crs.**CRSType**
Bases: *eoxserver.services.ows.wps.parameters.data_types.BaseType* (page 162)

CRS data-type. CRS are represented by the EPSG codes + 0 meaning the ImageCRC.

comparable = False

dtype
alias of builtins.int

classmethod encode(*value*)
Encode value to a Unicode string.

classmethod get_diff_dtype()
Get type of the difference of this type. E.g., *timedelta* for a *datetime*.

name = 'anyURI'

classmethod parse(*raw_value*)
Cast or parse input to its proper representation.

zero = None

eoxserver.services.ows.wps.parameters.data_types module

class eoxserver.services.ows.wps.parameters.data_types.**BaseType**
Bases: object[181]

Base literal data type class. This class defines the class interface.

[181] https://docs.python.org/3.6/library/functions.html#object

classmethod as_number(*value*)
convert to a number (e.g., duration)

comparable = True

dtype
alias of `builtins.bytes`

classmethod encode(*value*)
Encode value to a Unicode string.

classmethod get_diff_dtype()
Get type of the difference of this type. E.g., *timedelta* for a *datetime*.

name = None

classmethod parse(*raw_value*)
Cast or parse input to its proper representation.

classmethod sub(*value0*, *value1*)
subtract value0 - value1

zero = None

class eoxserver.services.ows.wps.parameters.data_types.**Boolean**
Bases: *eoxserver.services.ows.wps.parameters.data_types.BaseType* (page 162)

Boolean literal data type class.

classmethod as_number(*value*)
convert to a number (e.g., duration)

dtype
alias of `builtins.bool`

classmethod encode(*value*)
Encode value to a Unicode string.

name = 'boolean'

classmethod parse(*raw_value*)
Cast or parse input to its proper representation.

classmethod sub(*value0*, *value1*)
subtract value0 - value1

class eoxserver.services.ows.wps.parameters.data_types.**Date**
Bases: *eoxserver.services.ows.wps.parameters.data_types.BaseType* (page 162)

Date (*datetime.date*) literal data type class.

dtype
alias of `datetime.date`[182]

classmethod encode(*value*)
Encode value to a Unicode string.

classmethod get_diff_dtype()
Get type of the difference of this type. E.g., *timedelta* for a *datetime*.

name = 'date'

classmethod parse(*raw_value*)
Cast or parse input to its proper representation.

[182] https://docs.python.org/3.6/library/datetime.html#datetime.date

classmethod sub(*value0*, *value1*)
subtract value0 - value1

class eoxserver.services.ows.wps.parameters.data_types.**DateTime**
Bases: *eoxserver.services.ows.wps.parameters.data_types.BaseType* (page 162)

Date-time (*datetime.datetime*) literal data type class.

TZOffset(*name=None*)

UTC = <UTC>

dtype
alias of datetime.datetime[183]

classmethod encode(*value*)
Encode value to a Unicode string.

classmethod get_diff_dtype()
Get type of the difference of this type. E.g., *timedelta* for a *datetime*.

name = 'dateTime'

classmethod parse(*raw_value*)
Cast or parse input to its proper representation.

classmethod sub(*value0*, *value1*)
subtract value0 - value1

class eoxserver.services.ows.wps.parameters.data_types.**DateTimeTZAware**(*default_tz=<UTC>*, *target_tz=None*)
Bases: *eoxserver.services.ows.wps.parameters.data_types.DateTime* (page 164)

Time-zone aware date-time (*datetime.datetime*) literal data type class.

This data-type is a variant of the *DateTime* which assures that the parsed date-time is time-zone aware and optionally also converted to a common target time-zone.

The default time-zone applied to the unaware time-input is passed trough the constructor. By default the UTC time-zone is used. By default the target time-zone is set to None which means that the original time-zone is preserved.

Unlike the *DateTime* this class must be instantiated and it cannot be used directly as a data-type.

Constructor parameters: default_tz default time-zone target_tz optional target time-zone

encode(*value*)
Encode value to a Unicode string.

parse(*raw_value*)
Cast or parse input to its proper representation.

set_time_zone(*value*)
Make a date-time value time-zone aware by setting the default time-zone and convert the time-zone if the target time-zone is given.

class eoxserver.services.ows.wps.parameters.data_types.**Double**
Bases: *eoxserver.services.ows.wps.parameters.data_types.BaseType* (page 162)

Double precision float literal data type class.

classmethod as_number(*value*)
convert to a number (e.g., duration)

[183] https://docs.python.org/3.6/library/datetime.html#datetime.datetime

dtype
alias of builtins.float

classmethod encode(*value*)
Encode value to a Unicode string.

name = 'double'

classmethod sub(*value0*, *value1*)
subtract value0 - value1

zero = 0.0

class eoxserver.services.ows.wps.parameters.data_types.**Duration**
Bases: *eoxserver.services.ows.wps.parameters.data_types.BaseType* (page 162)

Duration (*datetime.timedelta*) literal data type class.

classmethod as_number(*value*)
convert to a number (e.g., duration)

dtype
alias of datetime.timedelta[184]

classmethod encode(*value*)
Encode value to a Unicode string.

name = 'duration'

classmethod parse(*raw_value*)
Cast or parse input to its proper representation.

classmethod sub(*value0*, *value1*)
subtract value0 - value1

zero = datetime.timedelta(0)

eoxserver.services.ows.wps.parameters.data_types.**FixedOffset**(*offset*, *name=None*)

class eoxserver.services.ows.wps.parameters.data_types.**Integer**
Bases: *eoxserver.services.ows.wps.parameters.data_types.BaseType* (page 162)

Integer literal data type class.

classmethod as_number(*value*)
convert to a number (e.g., duration)

dtype
alias of builtins.int

classmethod encode(*value*)
Encode value to a Unicode string.

name = 'integer'

classmethod sub(*value0*, *value1*)
subtract value0 - value1

zero = 0

class eoxserver.services.ows.wps.parameters.data_types.**String**
Bases: *eoxserver.services.ows.wps.parameters.data_types.BaseType* (page 162)

Unicode character string literal data type class.

[184] https://docs.python.org/3.6/library/datetime.html#datetime.timedelta

comparable = False

dtype
alias of builtins.str

classmethod encode(*value*)
Encode value to a Unicode string.

encoding = 'utf-8'

classmethod get_diff_dtype()
Get type of the difference of this type. E.g., *timedelta* for a *datetime*.

name = 'string'

classmethod parse(*raw_value*)
Cast or parse input to its proper representation.

class eoxserver.services.ows.wps.parameters.data_types.**Time**
Bases: *eoxserver.services.ows.wps.parameters.data_types.BaseType* (page 162)

Time (*datetime.time*) literal data type class.

dtype
alias of datetime.time[185]

classmethod encode(*value*)
Encode value to a Unicode string.

classmethod get_diff_dtype()
Get type of the difference of this type. E.g., *timedelta* for a *datetime*.

name = 'time'

classmethod parse(*raw_value*)
Cast or parse input to its proper representation.

classmethod sub(*value0*, *value1*)
subtract value0 - value1

eoxserver.services.ows.wps.parameters.formats module

class eoxserver.services.ows.wps.parameters.formats.**Format**(*encoder*, *mime_type*, *schema=None*, *is_text=False*, *is_xml=False*, *is_json=False*)
Bases: object[186]

Base complex data format.

Constructor parameters: encoder format's encoder object (defines the encoding) mime_type mime-type of the format schema optional schema of the document is_text optional boolean flag indicating text-based data

format.

is_xml optional boolean flag indicating XML-based format. The flag enables is_text flag.

is_json optional boolean flag indicating JSON-bases format. The flag enables is_text flag.

[185] https://docs.python.org/3.6/library/datetime.html#datetime.time
[186] https://docs.python.org/3.6/library/functions.html#object

allows_xml_embedding = False

decode(*file_in*, ***opt*)
Encoding generator.

encode(*file_in*, ***opt*)
Encoding generator.

encoding
Get the format encoding name.

class eoxserver.services.ows.wps.parameters.formats.**FormatBinaryBase64**(*mime_type='application/octet-stream'*)
Bases: *eoxserver.services.ows.wps.parameters.formats.Format* (page 166)

Base64 encoded binary complex data format.

allows_xml_embedding = True

class eoxserver.services.ows.wps.parameters.formats.**FormatBinaryRaw**(*mime_type='application/octet-stream'*)
Bases: *eoxserver.services.ows.wps.parameters.formats.Format* (page 166)

Raw binary complex data format.

allows_xml_embedding = False

class eoxserver.services.ows.wps.parameters.formats.**FormatJSON**(*mime_type='application/json'*, *schema=None*, *text_encoding='utf-8'*)
Bases: *eoxserver.services.ows.wps.parameters.formats.Format* (page 166)

JSON-based complex data format.

allows_xml_embedding = True

class eoxserver.services.ows.wps.parameters.formats.**FormatText**(*mime_type='text/plain'*, *schema=None*, *text_encoding='utf-8'*)
Bases: *eoxserver.services.ows.wps.parameters.formats.Format* (page 166)

Text-based complex data format.

allows_xml_embedding = True

class eoxserver.services.ows.wps.parameters.formats.**FormatXML**(*mime_type='application/xml'*, *schema=None*, *text_encoding='utf-8'*)
Bases: *eoxserver.services.ows.wps.parameters.formats.Format* (page 166)

XML-based complex data format.

allows_xml_embedding = True

eoxserver.services.ows.wps.parameters.inputs module

class eoxserver.services.ows.wps.parameters.inputs.**InputData**(*data*, *identifier*, *title=None*, *abstract=None*, *uom=None*, *crs=None*, *mime_type=None*, *encoding=None*, *schema=None*, *asurl=False*)

Bases: *eoxserver.services.ows.wps.parameters.base.ParamMetadata* (page 155)

Generic container for the raw data inputs. An instances of this class holds the inputs as decoded from various WPS requests before their validation and conversion to their configured data-type.

Constructor parameters: data unparsed (raw) data payload (byte string) identifier input item identifier title user defined title abstract user defined abstract uom input LiteralData UOM crs input BoundingBoxData CRS mime_type input ComplexData mime-type encoding input ComplexData encoding schema input ComplexData schema asurl indicates whether the decoded input comes from

a URL encoded request (KVP) or not.

class eoxserver.services.ows.wps.parameters.inputs.**InputReference**(*href*, *identifier*, *title=None*, *abstract=None*, *headers=None*, *body=None*, *method=None*, *mime_type=None*, *encoding=None*, *schema=None*, *body_href=None*)

Bases: *eoxserver.services.ows.wps.parameters.base.ParamMetadata* (page 155)

Input data reference class.

Constructor parameters: href input reference URL identifier input item identifier title user defined title abstract user defined abstract headers additional HTTP request headers body optional HTTP/POST request payload method reference method ('GET' or 'POST') mime_type reference ComplexData mime-type encoding reference ComplexData encoding schema reference ComplexData schema body_href optional HTTP/POST request payload reference URL

eoxserver.services.ows.wps.parameters.literaldata module

class eoxserver.services.ows.wps.parameters.literaldata.**LiteralData**(*identifier*, *dtype=<class 'eoxserver.services.ows.wps.para*, *uoms=None*, *default=None*, *allowed_values=None*, **args*, ***kwargs*)

Bases: *eoxserver.services.ows.wps.parameters.base.Parameter* (page 156)

Literal-data parameter class.

Constructor parameters: identifier identifier of the parameter used by the WPS service title optional human-readable name (defaults to identifier) abstract optional human-readable verbose description metadata optional metadata (title/URL dictionary) optional optional boolean flag indicating whether the input

parameter is optional or not

dtype optional data type of the parameter. String type `str` is set by default. For list of supported types see `LiteralData.SUPPORTED_TYPES`)

uoms optional sequence of the supported units default optional default input value. Presence of the

default value sets the parameter optional.

allowed_values optional restriction on the accepted values. By default any value of the given type is supported. The allowed value can be specified by an enumerated list (iterable) of values or by instance of one of the following classes: `AllowedAny`, `AllowedEnum`, `AllowedRange`, or `AllowedByReference`.

resolve_input_references Set this option to False not to resolve input references. By default the references are resolved (downloaded and parsed) transparently. If set to False the references must be handled by the process.

allowed_values
Allowed values object of the literal data object. (RO)

apply_uom(*value*, *uom*)
Convert value from the common base to the desired UOM.

check(*value*)
Check whether the value is allowed (True) or not (False).

default_uom
Get the default UOM.

dtype
Data type class of the literal data object. (RO)

encode(*value*, *uom=None*, *encoding=None*)
Encode the output value to its string representation.

The value is checked to match the defined allowed values restriction and the UOM conversion is applied.

Returns Unicode or byte-string if the encoding is given.

parse(*raw_value*, *uom=None*, *encoding='utf-8'*)
Parse the input value from its string representation.

The value is checked to match the defined allowed values restriction and the UOM conversion is applied.

Non-Unicode raw_data are converted to Unicode before parsing. Byte strings are decoded using the profited encoding (utf8 by default).

strip_uom(*value*, *uom*)
Convert value from the provided UOM to the common base.

uoms
Get all allowed UOMs.

verify(*value*)
Return the value if allowed or raise the ValueError exception.

eoxserver.services.ows.wps.parameters.response_form module

class eoxserver.services.ows.wps.parameters.response_form.**Output**(*identifier*, *title=None*, *abstract=None*, *uom=None*, *crs=None*, *mime_type=None*, *encoding=None*, *schema=None*, *as_reference=False*)
Bases: *eoxserver.services.ows.wps.parameters.base.ParamMetadata* (page 155)

Requested output definition.

Constructor parameters: identifier output identifier title output title (human-readable name) abstract output abstract (human-readable description) uom output LiteralData UOM crs output BoundingBox CRS mime_type output ComplexData mime-type encoding output ComplexData encoding schema output ComplexData schema as_reference boolean flag indicating whether the output should

passed as a reference op directly in the response.

class eoxserver.services.ows.wps.parameters.response_form.**RawDataOutput**(*output*)
Bases: *eoxserver.services.ows.wps.parameters.response_form.ResponseForm* (page 171)

Object representation of the raw output response.

Constructor parameters: output name of the requested output parameter

lineage = False

raw = True

status = False

store_response = False

class eoxserver.services.ows.wps.parameters.response_form.**ResponseDocument**(*lineage=False*, *status=False*, *store_response=False*)

Bases: *eoxserver.services.ows.wps.parameters.response_form.ResponseForm* (page 171)

Object representation of the (WPS Execute) response document.

Constructor parameters (meaning described in OGC 05-007r7, Table 50): lineage boolean flag, set to True to print the lineage status boolean flag, set to True to update status store_response boolean flag, set to True to store execute response

raw = False

class eoxserver.services.ows.wps.parameters.response_form.**ResponseForm**
Bases: collections.OrderedDict[187]

Response form defined as an ordered dictionary of the output definitions.

get_output(*identifier*)
Get an output for the given output identifier. An instance of the Output object is always returned.

set_output(*output*)
Set (insert) a new definition output.

eoxserver.services.ows.wps.parameters.units module

class eoxserver.services.ows.wps.parameters.units.**UnitLinear**(*name*, *scale*, *offset=0*)
Bases: *eoxserver.services.ows.wps.parameters.units.UnitOfMeasure* (page 171)

Simple unit of measure with linear conversion (scale and offset):

value_uom = (value_base - offset)/scale value_base = value_uom*scale + offset

Constructor parameters: name UOM name scale scale factor offset optional base offset (set to 0.0 by default)

Examples: For temperature conversions between the Fahrenheit scale (this UOM) and the Kelvin scale (base UOM) set scale to 5.0/9.0 and offset to 459.67*5.0/9.0 .

For simple distance conversions between kilometres (this UOM) and metres (base UOM) set scale factor to 1000.0 and offset to 0.0 .

apply(*value*)
Convert value from the common base to this unit.

strip(*value*)
Convert value of this unit to the common base.

class eoxserver.services.ows.wps.parameters.units.**UnitOfMeasure**(*name*)
Bases: object[188]

Base unit of measure class. The class defines conversion of input values in the given units to a common base unit and conversion of the output values from the common base unit to the this unit.

Constructor parameters: name UOM name

apply(*value*)
Convert value from the common base to this unit.

strip(*value*)
Convert value of this unit to the common base.

[187] https://docs.python.org/3.6/library/collections.html#collections.OrderedDict
[188] https://docs.python.org/3.6/library/functions.html#object

Module contents

class eoxserver.services.ows.wps.parameters.**Reference**(*path*, *href*, *mime_type=None*, *encoding=None*, *schema=None*, ***kwarg*)

Bases: object[189]

Output reference. An instance of this class defines a CommplexData output passed by a reference. The output must be stored in a file.

Constructor parameters: path path to the output file in the local file-system href public URL of the output reference mime_type output ComplexData mime-type encoding output ComplexData encoding schema output ComplexData schema

class eoxserver.services.ows.wps.parameters.**RequestParameter**(*request_parser=None*)

Bases: object[190]

Special input parameter extracting input from the request metadata. This might be used to pass information such as, e.g., HTTP headers or user authentication to the process like a regular input variable.

This class is the base class and it expected that *parse_request* method get overloaded by inheritance or by a function passed as an argument to the constructor.

parse_request(*request*)

Method extracting information from the Django HTTP request object.

eoxserver.services.ows.wps.parameters.**fix_parameter**(*name*, *prm*)

Expand short-hand definition of the parameter.

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

class eoxserver.services.ows.wps.v10.encoders.base.**WPS10BaseXMLEncoder**

Bases: *eoxserver.core.util.xmltools.XMLEncoder* (page 127)

Base class of the WPS 1.0 XML response encoders.

[189] https://docs.python.org/3.6/library/functions.html#object
[190] https://docs.python.org/3.6/library/functions.html#object

content_type = 'application/xml; charset=utf-8'

get_schema_locations()
Interface method. Returns a dict mapping namespace URIs to a network locations.

serialize(*tree*, ***kwargs*)
Serialize a XML tree to the pair (tuple) of the XML string and the content type.

eoxserver.services.ows.wps.v10.encoders.capabilities module

class eoxserver.services.ows.wps.v10.encoders.capabilities.**WPS10CapabilitiesXMLEncoder**
Bases: *eoxserver.services.ows.wps.v10.encoders.base.WPS10BaseXMLEncoder* (page 172)

WPS 1.0 Capabilities XML response encoder.

static encode_capabilities(*processes*)
Encode Capabilities XML document.

eoxserver.services.ows.wps.v10.encoders.execute_response module

class eoxserver.services.ows.wps.v10.encoders.execute_response.**WPS10ExecuteResponseXMLEncoc**

Bases: *eoxserver.services.ows.wps.v10.encoders.base.WPS10BaseXMLEncoder* (page 172)

WPS 1.0 ExecuteResponse XML response encoder.

encode_accepted()
Encode ProcessAccepted execute response.

encode_failed(*exception*)
Encode ProcessFailed execute response.

encode_paused(*progress=0*)
Encode ProcessPaused execute response.

encode_response(*results*)
Encode ProcessSucceeded execute response including the output data.

encode_started(*progress=0*, *message=None*)
Encode ProcessStarted execute response.

eoxserver.services.ows.wps.v10.encoders.execute_response_raw module

class eoxserver.services.ows.wps.v10.encoders.execute_response_raw.**ResultAlt**(*buf*, *content_type=None*, *filename=None*, *identifier=None*, *close=False*, *headers=None*)

Bases: *eoxserver.services.result.ResultItem* (page 195)

Alternative implementation of the result buffer. The object can be initialized with a byte-string, sequence or generator of byte-strings, or seekable file(-like) object.

chunked(*chunksize*)
Returns a chunk of the data, which has at most `chunksize` bytes.

data
Returns the "raw" data, usually as a string, buffer, memoryview, etc.

data_file
Returns the data as a Python file-like object.

class eoxserver.services.ows.wps.v10.encoders.execute_response_raw.**WPS10ExecuteResponseRawI**

Bases: `object`[191]

WPS 1.0 raw output Execute response encoder.

encode_response(*results*)
Pack the raw execute response.

static serialize(*result_items*, ***kwargs*)
Serialize the result items to the HTTP response object.

eoxserver.services.ows.wps.v10.encoders.parameters module

eoxserver.services.ows.wps.v10.encoders.parameters.**encode_input_descr**(*prm*)
Encode process description Input element.

eoxserver.services.ows.wps.v10.encoders.parameters.**encode_input_exec**(*prm*)
Encode common part of the execute response Input (data) element.

eoxserver.services.ows.wps.v10.encoders.parameters.**encode_output_def**(*outdef*)
Encode execute response Output (definition) element.

eoxserver.services.ows.wps.v10.encoders.parameters.**encode_output_descr**(*prm*)
Encode process description Output element.

eoxserver.services.ows.wps.v10.encoders.parameters.**encode_output_exec**(*prm*)
Encode common part of the execute response Output (data) element.

[191] https://docs.python.org/3.6/library/functions.html#object

eoxserver.services.ows.wps.v10.encoders.process_description module

class eoxserver.services.ows.wps.v10.encoders.process_description.**WPS10ProcessDescriptionsX**
Bases: *eoxserver.services.ows.wps.v10.encoders.base.WPS10BaseXMLEncoder* (page 172)

WPS 1.0 ProcessDescriptions XML response encoder.

static encode_process_descriptions(*processes*)
Encode the ProcessDescriptions XML document.

eoxserver.services.ows.wps.v10.encoders.process_description.**encode_process_brief**(*process*)
Encode a brief process description (Process element) of the Capabilities XML document.

eoxserver.services.ows.wps.v10.encoders.process_description.**encode_process_full**(*process*)
Encode a full process description (ProcessDescription element) of the ProcessDescriptions XML document.

Module contents

Submodules

eoxserver.services.ows.wps.v10.describeprocess module

class eoxserver.services.ows.wps.v10.describeprocess.**WPS10DescribeProcessHandler**
Bases: object[192]

WPS 1.0 DescribeProcess service handler.

static get_decoder(*request*)
Get the WPS request decoder.

handle(*request*)
Handle HTTP request.

methods = ['GET', 'POST']

request = 'DescribeProcess'

service = 'WPS'

versions = ('1.0.0',)

class eoxserver.services.ows.wps.v10.describeprocess.**WPS10DescribeProcessKVPDecoder**(*params*)
Bases: *eoxserver.core.decoders.kvp.Decoder* (page 118)

WPS 1.0 DescribeProcess HTTP/GET KVP request decoder.

identifiers
Property getter function.

class eoxserver.services.ows.wps.v10.describeprocess.**WPS10DescribeProcessXMLDecoder**(*tree*)
Bases: *eoxserver.core.decoders.xml.Decoder* (page 119)

WPS 1.0 DescribeProcess HTTP/POST XML request decoder.

identifiers
Property getter function.

namespaces = {'ows': 'http://www.opengis.net/ows/1.1', 'wps': 'http://www.opengis.net

[192] https://docs.python.org/3.6/library/functions.html#object

eoxserver.services.ows.wps.v10.exceptionhandler module

class eoxserver.services.ows.wps.v10.exceptionhandler.**WPS10ExceptionHandler**(**args*)
Bases: *eoxserver.core.component.Component* (page 128)

WPS 1.0 exception handler.

handle_exception(*request*, *exception*)
Handle exception.

request = None

service = 'WPS'

versions = ('1.0.0', '1.0')

eoxserver.services.ows.wps.v10.execute module

class eoxserver.services.ows.wps.v10.execute.**WPS10ExecuteHandler**
Bases: object[193]

WPS 1.0 Execute service handler.

get_async_backend()
Get available asynchronous back-end matched by the service version.

static get_decoder(*request*)
Get request decoder matching the request format.

get_process(*identifier*)
Get process component matched by the identifier.

handle(*request*)
Request handler.

methods = ['GET', 'POST']

request = 'Execute'

service = 'WPS'

versions = ('1.0.0',)

eoxserver.services.ows.wps.v10.execute_decoder_kvp module

class eoxserver.services.ows.wps.v10.execute_decoder_kvp.**WPS10ExecuteKVPDecoder**(*params*)
Bases: *eoxserver.core.decoders.kvp.Decoder* (page 118)

WPS 1.0 Execute HTTP/GET KVP request decoder.

identifier
Property getter function.

inputs
Property getter function.

lineage
Property getter function.

[193] https://docs.python.org/3.6/library/functions.html#object

outputs
Property getter function.

raw_response
Property getter function.

response_form
Get response unified form parsed either from ResponseDocument or RawDataOutput parameters.

status
Property getter function.

store_response
Property getter function.

eoxserver.services.ows.wps.v10.execute_decoder_kvp.**parse_query_string**(*query_string*)
Parse URL query string preserving the URL-encoded DataInputs, ResponseDocument, and RawDataOutput WPS Execute parameters. Note that the standard parser URL-decodes the parameter values and, in cases when, e.g., a data input contains an percent-encoded separator ('%40' vs. '@') the encoded and non-encoded delimiters cannot be distinguished ('@' vs. '@') and the correct parsing cannot be guaranteed.

eoxserver.services.ows.wps.v10.execute_decoder_xml module

class eoxserver.services.ows.wps.v10.execute_decoder_xml.**WPS10ExecuteXMLDecoder**(*tree*)
Bases: *eoxserver.core.decoders.xml.Decoder* (page 119)

WPS 1.0 POST/XML Execute request decoder class.

identifier
Property getter function.

inputs
Get the raw data inputs as a dictionary.

namespaces = {'ows': 'http://www.opengis.net/ows/1.1', 'wps': 'http://www.opengis.ne

response_form
Get the unified response form object.

eoxserver.services.ows.wps.v10.getcapabilities module

class eoxserver.services.ows.wps.v10.getcapabilities.**WPS10GetCapabilitiesHandler**
Bases: object[194]

WPS 1.0 GetCapabilities service handler.

handle(*request*)
Handle HTTP request.

methods = ['GET', 'POST']

request = 'GetCapabilities'

service = 'WPS'

versions = ('1.0.0',)

[194] https://docs.python.org/3.6/library/functions.html#object

class eoxserver.services.ows.wps.v10.getcapabilities.**WPS10GetCapabilitiesKVPDecoder**(*params*)
Bases: *eoxserver.core.decoders.kvp.Decoder* (page 118)

WPS 1.0 GetCapabilities HTTP/GET KVP request decoder.

language
Property getter function.

class eoxserver.services.ows.wps.v10.getcapabilities.**WPS10GetCapabilitiesXMLDecoder**(*tree*)
Bases: *eoxserver.core.decoders.xml.Decoder* (page 119)

WPS 1.0 DescribeProcess HTTP/POST XML request decoder.

language
Property getter function.

namespaces = {'ows': 'http://www.opengis.net/ows/1.1', 'wps': 'http://www.opengis.ne

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 179)

exception eoxserver.services.ows.wps.exceptions.**FileSizeExceeded**(*message*, *locator*)
Bases: *eoxserver.services.ows.wps.exceptions.OWS10Exception* (page 179)

exception eoxserver.services.ows.wps.exceptions.**InvalidInputError**(*input_id*)
Bases: *eoxserver.services.ows.wps.exceptions.InvalidParameterValue* (page 178)

exception eoxserver.services.ows.wps.exceptions.**InvalidInputReferenceError**(*input_id*, *message=''*)
Bases: *eoxserver.services.ows.wps.exceptions.InvalidParameterValue* (page 178)

exception eoxserver.services.ows.wps.exceptions.**InvalidInputValueError**(*input_id*, *message=''*)
Bases: *eoxserver.services.ows.wps.exceptions.InvalidParameterValue* (page 178)

exception eoxserver.services.ows.wps.exceptions.**InvalidOutputDefError**(*output_id*, *message=''*)
Bases: *eoxserver.services.ows.wps.exceptions.InvalidParameterValue* (page 178)

exception eoxserver.services.ows.wps.exceptions.**InvalidOutputError**(*output_id*)
Bases: *eoxserver.services.ows.wps.exceptions.InvalidParameterValue* (page 178)

exception eoxserver.services.ows.wps.exceptions.**InvalidOutputValueError**(*output_id*, *message=''*)
Bases: *eoxserver.services.ows.wps.exceptions.NoApplicableCode* (page 179)

exception eoxserver.services.ows.wps.exceptions.**InvalidParameterValue**(*message*, *locator*)
Bases: *eoxserver.services.ows.wps.exceptions.OWS10Exception* (page 179)

exception eoxserver.services.ows.wps.exceptions.**MissingParameterValue**(*message*, *locator*)
Bases: *eoxserver.services.ows.wps.exceptions.OWS10Exception* (page 179)

exception eoxserver.services.ows.wps.exceptions.**MissingRequiredInputError**(*input_id*)
Bases: *eoxserver.services.ows.wps.exceptions.InvalidParameterValue* (page 178)

exception eoxserver.services.ows.wps.exceptions.**NoApplicableCode**(*message*, *locator=None*)
Bases: *eoxserver.services.ows.wps.exceptions.OWS10Exception* (page 179)

http_status_code = 500

exception eoxserver.services.ows.wps.exceptions.**NoSuchProcessError**(*identifier*)
Bases: *eoxserver.services.ows.wps.exceptions.InvalidParameterValue* (page 178)

exception eoxserver.services.ows.wps.exceptions.**NotEnoughStorage**(*message*)
Bases: *eoxserver.services.ows.wps.exceptions.OWS10Exception* (page 179)

http_status_code = 507

exception eoxserver.services.ows.wps.exceptions.**OWS10Exception**(*code*, *locator*, *message*)
Bases: Exception[195]

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 179)

http_status_code = 503

exception eoxserver.services.ows.wps.exceptions.**StorageNotSupported**(*message*)
Bases: *eoxserver.services.ows.wps.exceptions.OWS10Exception* (page 179)

exception eoxserver.services.ows.wps.exceptions.**VersionNegotiationFailed**(*message*, *locator*)
Bases: *eoxserver.services.ows.wps.exceptions.OWS10Exception* (page 179)

eoxserver.services.ows.wps.interfaces module

class eoxserver.services.ows.wps.interfaces.**AsyncBackendInterface**
Bases: object[196]

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.

[195] https://docs.python.org/3.6/library/exceptions.html#Exception
[196] https://docs.python.org/3.6/library/functions.html#object

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`[197]

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.

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.

[197] https://docs.python.org/3.6/library/functions.html#object

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

class eoxserver.services.ows.wps.test_allowed_values.**BaseTestMixin**
Bases: `object`[198]

test()

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedAny**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumDate**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumDate2**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumDateTime**(*methodName='runT*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

[198] https://docs.python.org/3.6/library/functions.html#object

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumDateTime2**(*methodName='ru*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumDuration**(*methodName='runT*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumDuration2**(*methodName='ru*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumFloat**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumFloat2**(*methodName='runTest*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumFloat3**(*methodName='runTest*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumInt**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumInt2**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumString**(*methodName='runTest*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumString2**(*methodName='runTe*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumString3**(*methodName='runTe*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumTime**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedEnumTime2**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeCollectionFloat**(*methodI*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDateClosed**(*methodName='*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDateClosedOpen**(*methodNa*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDateOpen**(*methodName='ru*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDateOpenClosed**(*methodNa*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDateTime**(*methodName='ru*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDiscrDate**(*methodName='r*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDiscrDateTime**(*methodNa*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDiscrDuration**(*methodNa*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDiscrFloat**(*methodName=*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDiscrInt**(*methodName='ru*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDiscrTime**(*methodName='r*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeDuration**(*methodName='ru*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeFloat**(*methodName='runTest*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeFloat2**(*methodName='runTe*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeFloat3**(*methodName='runTe*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeFloatClosed**(*methodName=*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeFloatClosedOpen**(*methodl*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeFloatOpen**(*methodName='r*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeFloatOpenClosed**(*methodl*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeInt**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeIntClosed**(*methodName='r*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeUnboundMax**(*methodName=*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_allowed_values.**TestAllowedRangeUnboundMin**(*methodName='*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_allowed_values.BaseTestMixin* (page 181)

setUp()
Hook method for setting up the test fixture before exercising it.

eoxserver.services.ows.wps.test_data_types module

class eoxserver.services.ows.wps.test_data_types.**BaseTestMixin**
Bases: object[199]

testEncodeFail()

testEncodeOK()

testGeneral()

testParseFail()

testParseOK()

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeBool**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeCRS**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeDate**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeDateTime**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186), *eoxserver.services.ows.wps.test_data_types.TimeZoneTestMixin* (page 187)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeDateTimeTZAware**(*methodName='runT*
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186), *eoxserver.services.ows.wps.test_data_types.TimeZoneTestMixin* (page 187)

[199] https://docs.python.org/3.6/library/functions.html#object

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeDateTimeTZAwareWithTZConversio**
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186), *eoxserver.services.ows.wps.test_data_types.TimeZoneTestMixin* (page 187)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeDuration**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeFloat**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeInt**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeString**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TestDataTypeTime**(*methodName='runTest'*)
Bases: unittest.case.TestCase, *eoxserver.services.ows.wps.test_data_types.BaseTestMixin* (page 186)

setUp()
Hook method for setting up the test fixture before exercising it.

class eoxserver.services.ows.wps.test_data_types.**TimeZoneTestMixin**
Bases: object[200]

testParseTimeZone()

Module contents

Submodules

[200] https://docs.python.org/3.6/library/functions.html#object

eoxserver.services.ows.component module

class eoxserver.services.ows.component.**OptionsRequestHandler**
Bases: object[201]

Dummy request handler class to respond to HTTP OPTIONS requests.

handle(*request*)

class eoxserver.services.ows.component.**ServiceComponent**(**args*, ***kwargs*)
Bases: *eoxserver.core.component.Component* (page 128)

exception_handlers
List of components that implement *eoxserver.services.ows.interfaces.ExceptionHandlerInterface* (page 189)

get_service_handlers
List of components that implement *eoxserver.services.ows.interfaces.GetServiceHandlerInterface* (page 190)

post_service_handlers
List of components that implement *eoxserver.services.ows.interfaces.PostServiceHandlerInterface* (page 190)

query_exception_handler(*request*)

query_service_handler(*request*)
Tries to find the correct service handler for a given request. The request `method` can either be "POST" (in which case the request body is parsed as XML) or "GET" (in which case the request is parsed as "KVP").

If necessary a version negotiation is conducted, following OWS guidelines.

Parameters **request** – a `Django HttpRequest`[202] object

Returns the request handler component for the given request

Raises

- ***ServiceNotSupportedException*** (page 193) – if the service is not supported by any component
- ***VersionNotSupportedException*** (page 193) – if the specified version is not supported
- ***OperationNotSupportedException*** (page 193) – if the specified request operation is not supported

query_service_handlers(*service=None*, *versions=None*, *request=None*, *method=None*)
Query the service handler components, filtering optionally by `service`, `versions`, `request` or `method`.

service_handlers
List of components that implement *eoxserver.services.ows.interfaces.ServiceHandlerInterface* (page 190)

version_negotiation(*handlers*, *accepted_versions=None*)

version_negotiation_handlers
List of components that implement *eoxserver.services.ows.interfaces.VersionNegotiationInterface* (page 190)

[201] https://docs.python.org/3.6/library/functions.html#object
[202] https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.HttpRequest

eoxserver.services.ows.component.**filter_handlers**(*handlers*, *service=None*, *versions=None*, *request=None*)
Utility function to filter the given OWS service handlers by their attributes 'service', 'versions' and 'request'.

eoxserver.services.ows.component.**handler_supports_service**(*handler*, *service=None*)
Convenience method to check whether or not a handler supports a service.

eoxserver.services.ows.component.**sort_handlers**(*handlers*, *ascending=True*)

eoxserver.services.ows.decoders module

class eoxserver.services.ows.decoders.**OWSCommonKVPDecoder**(*params*)
Bases: *eoxserver.core.decoders.kvp.Decoder* (page 118)

acceptversions
Property getter function.

request
Property getter function.

service
Property getter function.

version
Property getter function.

class eoxserver.services.ows.decoders.**OWSCommonXMLDecoder**(*tree*)
Bases: *eoxserver.core.decoders.xml.Decoder* (page 119)

acceptversions
Property getter function.

namespaces = {'ows10': 'http://www.opengis.net/ows/1.0', 'ows11': 'http://www.opengi

request
Property getter function.

service
Property getter function.

version
Property getter function.

eoxserver.services.ows.decoders.**get_decoder**(*request*)
Convenience function to return the right OWS Common request deocder for the given *django.http.HttpRequest*.

eoxserver.services.ows.interfaces module

class eoxserver.services.ows.interfaces.**ExceptionHandlerInterface**
Bases: object[203]

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.

[203] https://docs.python.org/3.6/library/functions.html#object

request
The supported request method.

service
The 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 alse 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 190)

Interface for service handlers that support HTTP GET requests.

class eoxserver.services.ows.interfaces.**PostServiceHandlerInterface**
Bases: *eoxserver.services.ows.interfaces.ServiceHandlerInterface* (page 190)

Interface for service handlers that support HTTP POST requests.

class eoxserver.services.ows.interfaces.**ServiceHandlerInterface**
Bases: `object`[204]

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.

service
The 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 alse 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 190)

Interface for handlers that contribute to version negotiation.

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`[205]

[204] https://docs.python.org/3.6/library/functions.html#object
[205] https://docs.python.org/3.6/library/functions.html#object

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 compareable.

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: Exception[206]

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 `mgs` and the list of the accepted HTTP methods `allowed_methods`.

exception eoxserver.services.exceptions.**InterpolationMethodNotSupportedException**

Bases: Exception[207]

This exception indicates a not supported interpolation method.

code = 'InterpolationMethodNotSupported'

locator = 'interpolation'

exception eoxserver.services.exceptions.**InvalidAxisLabelException**(*axis_label*)

Bases: Exception[208]

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: Exception[209]

Error in RangeSubsetting for illegal intervals.

code = 'InvalidFieldSequence'

exception eoxserver.services.exceptions.**InvalidOutputCrsException**

Bases: Exception[210]

[206] https://docs.python.org/3.6/library/exceptions.html#Exception
[207] https://docs.python.org/3.6/library/exceptions.html#Exception
[208] https://docs.python.org/3.6/library/exceptions.html#Exception
[209] https://docs.python.org/3.6/library/exceptions.html#Exception
[210] https://docs.python.org/3.6/library/exceptions.html#Exception

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: Exception[211]

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: Exception[212]

Error in ScaleExtent operations

code = 'InvalidExtent'

exception eoxserver.services.exceptions.**InvalidScaleFactorException**(*scalefactor*)

Bases: Exception[213]

Error in ScaleFactor and ScaleAxis operations

code = 'InvalidScaleFactor'

exception eoxserver.services.exceptions.**InvalidSubsettingCrsException**

Bases: Exception[214]

This exception indicates an invalid WCS 2.0 subsettingCrs parameter was submitted.

code = 'SubsettingCrs-NotSupported'

locator = 'subsettingCrs'

exception eoxserver.services.exceptions.**InvalidSubsettingException**

Bases: Exception[215]

This exception indicates an invalid WCS 2.0 subsetting parameter was submitted.

code = 'InvalidSubsetting'

locator = 'subset'

exception eoxserver.services.exceptions.**LocatorListException**(*items*)

Bases: Exception[216]

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.

[211] https://docs.python.org/3.6/library/exceptions.html#Exception
[212] https://docs.python.org/3.6/library/exceptions.html#Exception
[213] https://docs.python.org/3.6/library/exceptions.html#Exception
[214] https://docs.python.org/3.6/library/exceptions.html#Exception
[215] https://docs.python.org/3.6/library/exceptions.html#Exception
[216] https://docs.python.org/3.6/library/exceptions.html#Exception

exception eoxserver.services.exceptions.**NoSuchCoverageException**(*items*)
Bases: *eoxserver.services.exceptions.LocatorListException* (page 192)

This exception indicates that the requested coverage(s) do not exist.

code = 'NoSuchCoverage'

exception eoxserver.services.exceptions.**NoSuchDatasetSeriesOrCoverageException**(*items*)
Bases: *eoxserver.services.exceptions.LocatorListException* (page 192)

This exception indicates that the requested coverage(s) or dataset series do not exist.

code = 'NoSuchDatasetSeriesOrCoverage'

exception eoxserver.services.exceptions.**NoSuchFieldException**(*msg*, *locator*)
Bases: Exception[217]

Error in RangeSubsetting when band does not exist.

code = 'NoSuchField'

exception eoxserver.services.exceptions.**OperationNotSupportedException**(*message*, *operation=None*)
Bases: Exception[218]

Exception to be thrown when some operations are not supported or disabled.

code = 'OperationNotSupported'

locator

exception eoxserver.services.exceptions.**RenderException**(*message*, *locator*, *is_parameter=True*)
Bases: Exception[219]

Rendering related exception.

code

exception eoxserver.services.exceptions.**ScaleAxisUndefinedException**(*axis*)
Bases: Exception[220]

Error in all scaling operations involving an axis

code = 'ScaleAxisUndefined'

exception eoxserver.services.exceptions.**ServiceNotSupportedException**(*service*)
Bases: *eoxserver.services.exceptions.OperationNotSupportedException* (page 193)

Exception to be thrown when a specific OWS service is not enabled.

exception eoxserver.services.exceptions.**VersionNegotiationException**
Bases: Exception[221]

This exception indicates that version negotiation fails. Such errors can happen with OWS 2.0 compliant "new-style" version negotation.

code = 'VersionNegotiationFailed'

[217] https://docs.python.org/3.6/library/exceptions.html#Exception
[218] https://docs.python.org/3.6/library/exceptions.html#Exception
[219] https://docs.python.org/3.6/library/exceptions.html#Exception
[220] https://docs.python.org/3.6/library/exceptions.html#Exception
[221] https://docs.python.org/3.6/library/exceptions.html#Exception

exception eoxserver.services.exceptions.**VersionNotSupportedException**(*service*, *version*)

Bases: Exception[222]

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*, *sections=None*, *accept_languages=None*, *accept_formats=None*, *updatesequence=None*, *request=None*)

Bases: object[223]

accept_formats

accept_languages

coverages

request

sections

updatesequence

version

class eoxserver.services.parameters.**RenderParameters**

Bases: object[224]

Abstract base class for render parameters

class eoxserver.services.parameters.**VersionedParams**(*version*)

Bases: object[225]

version

eoxserver.services.result module

class eoxserver.services.result.**ResultBuffer**(*buf*, *content_type=None*, *filename=None*, *identifier=None*)

Bases: *eoxserver.services.result.ResultItem* (page 195)

Class for results that are actually a subset of a larger context. Usually a buffer.

[222] https://docs.python.org/3.6/library/exceptions.html#Exception
[223] https://docs.python.org/3.6/library/functions.html#object
[224] https://docs.python.org/3.6/library/functions.html#object
[225] https://docs.python.org/3.6/library/functions.html#object

chunked(*chunksize*)
Returns a chunk of the data, which has at most `chunksize` bytes.

data
Returns the "raw" data, usually as a string, buffer, memoryview, etc.

class eoxserver.services.result.**ResultFile**(*path*, *content_type=None*, *filename=None*, *identifier=None*)
Bases: *eoxserver.services.result.ResultItem* (page 195)

Class for results that wrap physical files on the disc.

chunked(*chunksize*)
Returns a chunk of the data, which has at most `chunksize` bytes.

data
Returns the "raw" data, usually as a string, buffer, memoryview, etc.

data_file
Returns the data as a Python file-like object.

delete()
Cleanup any associated files, allocated memory, etc.

class eoxserver.services.result.**ResultItem**(*content_type=None*, *filename=None*, *identifier=None*)
Bases: `object`[226]

Base class (or interface) for result items of a result set.

Parameters

- **content_type** – the content type of the result item. in HTTP this will be translated to the `Content-Type` header
- **filename** – the filename of the result item.
- **identifier** – the identifier of the result item. translated to `Content-Id` HTTP header

chunked(*chunksize*)
Returns a chunk of the data, which has at most `chunksize` bytes.

content_type
Reterns a binary value of content-type if it is a string.

data
Returns the "raw" data, usually as a string, buffer, memoryview, etc.

data_file
Returns the data as a Python file-like object.

delete()
Cleanup any associated files, allocated memory, etc.

size

eoxserver.services.result.**get_content_type**(*result_set*)
Returns the content type of a result set. If only one item is included its content type is used, otherwise the constant "multipart/related".

eoxserver.services.result.**get_headers**(*result_item*)
Yields content headers, if they are set in the result item.

[226] https://docs.python.org/3.6/library/functions.html#object

eoxserver.services.result.**get_payload_size**(*result_set*, *boundary*)
Calculate the size of the result set and all entailed result items plus headers.

eoxserver.services.result.**parse_headers**(*headers*)
Convenience function to read the "Content-Type", "Content-Disposition" and "Content-Id" headers.

Parameters **headers** – the raw header dict[227]

eoxserver.services.result.**result_set_from_raw_data**(*data*)
Create a result set from raw HTTP data. This can either be a single or a multipart string. It returns a list containing objects of the *ResultBuffer* (page 194) type that reference substrings of the given data.

Parameters **data** – the raw byte data

Returns a result set: a list containing *ResultBuffer* (page 194)

eoxserver.services.result.**to_http_response**(*result_set*, *response_type=<class 'django.http.response.HttpResponse'>*, *boundary=None*)
Returns a response for a given result set. The response_type is the class to be used. It must be capable to work with iterators. This function is also responsible to delete any temporary files and buffers of the result_set.

Parameters

- **result_set** – an iterable of objects following the *ResultItem* (page 195) interface
- **response_type** – the response type class to use; defaults to HttpResponse[228]. For streaming responses use StreamingHttpResponse[229]
- **boundary** – the multipart boundary; if omitted a UUID hex string is computed and used

Returns a response object of the desired type

eoxserver.services.subset module

eoxserver.services.urls module

eoxserver.services.urls.**get_http_service_url**(*request=None*)
Returns the URL the OWS view is available under. If a django.http.HttpRequest[230] is passed, an absolute URL is constructed with the request information.

eoxserver.services.views module

This model contains Django views for the EOxServer software. Its main function is *ows()* (page 196) which handles all incoming OWS requests

eoxserver.services.views.**ows**(*request*)
Main entry point for OWS requests against EOxServer. It uses the *ServiceComponent* (page 188) to dynamically determine the handler component for this request.

If an exception occurs during the handling of the request, an exception handler component is determined and dispatched.

[227] https://docs.python.org/3.6/library/stdtypes.html#dict
[228] https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.HttpResponse
[229] https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.StreamingHttpResponse
[230] https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.HttpRequest

Any response of the service handler and exception handler is transformed to a django `HttpResponse`[231] to adhere the required interface.

Module contents

4.1.7 eoxserver.testing package

Submodules

eoxserver.testing.xcomp module

Simple XML documets' comparator.

exception eoxserver.testing.xcomp.**XMLError**
Bases: `Exception`[232]

XML base error error

exception eoxserver.testing.xcomp.**XMLMismatchError**
Bases: *eoxserver.testing.xcomp.XMLError* (page 197)

XML mismatch error

exception eoxserver.testing.xcomp.**XMLParseError**
Bases: *eoxserver.testing.xcomp.XMLError* (page 197)

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

4.1.8 eoxserver.webclient package

Submodules

eoxserver.webclient.models module

eoxserver.webclient.urls module

eoxserver.webclient.views module

[231] https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.HttpResponse
[232] https://docs.python.org/3.6/library/exceptions.html#Exception

Module contents

4.2 Submodules

4.3 eoxserver.views module

eoxserver.views.**index**(*request*)

4.4 Module contents

eoxserver.**get_version**()

CHAPTER 5

License

5.1 EOxServer Open License

EOxServer Open License
Version 1, 8 June 2011

CHAPTER 6

Credits

[233]

Work on EOxServer has been partly funded by the European Space Agency (ESA)[234] in the frame of the HMA-FO[235] and O3S[236] projects.

[233] http://rssportal.esa.int/tiki-index.php?page=Open%20Software
[234] http://www.esa.int/esaMI/ESRIN_SITE/
[235] http://wiki.services.eoportal.org/tiki-index.php?page=HMA-FO
[236] http://wiki.services.eoportal.org/tiki-index.php?page=O3S

Index

A

B

C

D

E

F

G

K

L

M

N

O

P

Q

R

S

T

W

X

Z