
Stetl Documentation

Release 1.2-dev

Just van den Broecke

Apr 04, 2018

Contents

1	Intro	3
2	Installation	5
3	Background	9
4	Using Stetl	13
5	Cases	23
6	API and Code	25
7	Contact	55
8	Links	57
9	Presentations	59
10	Stetl Projects/Cases	61
11	Tools	63
12	Other	65
13	Indices and tables	67
	Python Module Index	69

Stetl, Streaming ETL, is an open source (GNU GPL) toolkit for the transformation (ETL) of geospatial data. Stetl is based on existing ETL tools like GDAL/OGR and XSLT. Stetl processing is driven from a configuration (.ini) file. Stetl is written in Python and in particular suited for processing GML.

This is the documentation of the Stetl toolkit. The code is on GitHub: <https://github.com/geopython/stetl>. Since July 2016 the project is a proud member of the [GeoPython GitHub organization](#).

See an [introductory Stetl presentation on Slideshare](#).

This is document version 1.2-dev generated on Apr 03, 2018.

Contents:

Stetl, streaming ETL, pronounced “staedl”, is a lightweight ETL-framework for the conversion of rich (such as GML) geospatial data. Stetl is Open Source (GNU GPL v3).

Read a 5-minute introduction here: <http://www.slideshare.net/justb4/5-minute-intro-to-setl> and a longer presentation here: <http://www.slideshare.net/justb4/geospatial-etl-with-stetl-geopython-2016>. Plus a presentation of Stetl for use in INSPIRE transformation: <http://www.slideshare.net/justb4/2-stetlinspiretransformv1> with even a video recording: <https://www.youtube.com/watch?v=vjdpYBm4AaM>

Stetl originated in the [INSPIRE-FOSS project](#) and was originally created by [Just van den Broecke](#). Subsequently, Stetl evolved into a wider use transforming Dutch GML-based datasets such as Top10NL, IMGEO/BGT (Large Scale Topography) and IMKAD/BRK (Kadastral Data). Therefore Stetl now has a repository of its own at [GitHub](#).

Stetl basically glues together existing parsing and transformation tools like [GDAL/OGR \(ogr2ogr\)](#) and [XSLT](#). By using native tools like [libxml2](#) and [libxslt](#) (via [Python lxml](#)) Stetl is speed-optimized.

Stetl has (currently) no GUI. There are powerful Open Source ETL tools like [GeoKettle](#) and Talend Geospatial with a GUI. Check these out. But some of us would like to stay close to the commandline, be Pythonic and reuse existing tools ‘close to the iron’.

So why and when to use Stetl:

- when ogr2ogr or XSLT alone cannot do the job
- when having to deal with complex GML as source or destination
- when you want to use simple command-line tooling or (Python) program integrations
- when you need speed
- when you are a *Pythonista*

Stetl is in particularly useful for INSPIRE-related transformations and other complex GML-related ETL.

Stetl was presented at FOSS4G 2013 in Nottingham, see <http://2013.foss4g.org/conf/programme/presentations/156> and the slides: <http://www.slideshare.net/justb4/stetl-foss4g20131024v1>

Installation

Stetl currently only runs with Python 2 (2.7+). [Work is underway](#) for Python3 support.

Easiest is to first install the Stetl-dependencies (see below) and then install and maintain Stetl on your system as a Python package (pip is preferred).

```
(sudo) pip install stetl
or
easy_install stetl
```

Alternatively you can download Stetl from Github: by cloning (preferred) or downloading: <https://github.com/geopython/stetl/archive/master.zip> and then install locally

```
(sudo) python setup.py install
```

Try the examples first. This should work on Linuxes and Mac OSX.

Windows installation may be more involved depending on your local Python setup. Platform-specific installations below.

You may also want to download the complete .tar.gz distro from PyPi: <https://pypi.python.org/pypi/Stetl> . This includes the examples and tests.

Docker

Since version 1.0.9 Stetl also can be installed and run via [Docker](#). See *Install with Docker* below.

Debian/Ubuntu

Thanks to Bas Couwenberg, work is performed to provide Stetl as Debian packages on both Debian and Ubuntu, see details: <https://packages.debian.org/search?keywords=stetl> (Debian) and <https://launchpad.net/ubuntu/+source/python-stetl> (Ubuntu, Xenial and later). Stetl is split into 2 packages `python-stetl`, the Python framework and `stetl` the command line utility. NB the versions of these packages may be older than when installing Stetl via *pip* from PyPi or directly from GitHub. Always check this first.

2.1 Dependencies

Stetl depends on the following Python packages:

- GDAL (v2+) bindings for Python
- psycopg2 (PostgreSQL client)
- lxml
- Jinja2 templating

GDAL Python binding requires the native GDAL/OGR libs and tools (version 2+) to be installed.

lxml <http://lxml.de/installation.html> requires the native (C) libraries:

- libxslt (required by lxml)
- libxml2 with Python bindings (required by lxml)

When using the Jinja2 templating filter, `Jinja2TemplatingFilter`, see <http://jinja.pocoo.org>:

- Python Jinja2 package

Platform-specific guidelines for dependencies follow next.

2.1.1 Linux

For Debian-based distro's like Ubuntu and Debian itself, most packages should be able to be installed via `apt-get`.

Tip: to get latest versions of GDAL and other Open Source geospatial software, best is to add the [UbuntuGIS Repository](#). Below a setup that works in Ubuntu 16.04 Xenial using Debian/Ubuntu packages. In some cases you may choose to install the same packages via `pip` to have more recent versions like for `lxml`.

- Python dependencies:

```
apt-get install python-setuptools
apt-get install python-dev
apt-get install python-pip
pip install --upgrade pip
```

- `libxml2/libxslt` libs are usually already installed. Together with Python `lxml`, the total install for `lxml` is:

```
apt-get install python-libxml2
apt-get install python-libxslt1
apt-get install libxml2-dev libxslt1-dev lib32z1-dev
apt-get install python-lxml
```

- GDAL (<http://gdal.org>) version 2+ with Python bindings:

```
# Add UbuntuGIS repo to get latest GDAL, at least v2 on Ubuntu 16.04, Xenial.
add-apt-repository ppa:ubuntugis/ubuntugis-unstable
apt-get update
apt-get install gdal-bin
gdalinfo --version
# should show something like: GDAL 2.2.1, released 2017/06/23

apt-get install python-gdal
```

- the PostgreSQL client library for Python `psycopg2`:

```
apt-get install python-psycpg2
```

- for Jinja2:

```
apt-get install python-jinja2
```

2.1.2 Mac OSX

Dependencies can best be installed via [Homebrew](#).

2.1.3 Windows

Best is to install GDAL and python using the OSGeo4W Installer from <http://trac.osgeo.org/osgeo4w>.

- Download and run the OSGeo4W Installer
- Choose Advanced Install
- On the Select Packages page expand Commandline_Uutilities and Select from the list gdal and python
- (psycpg2??)
- Install easy_install to allow you to install lxml
- Download the ez_setup.py script
- Open the OSGeo4W Shell (Start > Programs > OSGeo4W > OSGeo4W > OSGeo4W Shell)
- Change to the folder that you downloaded ez_setup.py to (if you downloaded to C:Temp then run cd C:Temp)
- Install easy_install by running python ez_setup.py
- To install lxml with easy_install run easy_install lxml

Only Psycpg2 needs explicit installation. Many install via: <http://www.stickpeople.com/projects/python/win-psycpg>. Once the above has been installed you should have everything required to run Stetl.

Alternatively you may use Portable GIS. Still you will need to manually install psycpg2. See <http://www.archaeogeek.com/portable-gis.html> for details.

2.2 Test Installation

If you installed via Python 'pip' you can check if you run the latest version

```
stetl -h
```

You should get meaningful output like

```
2013-09-16 18:25:12,093 util INFO running with lxml.etree, good!
2013-09-16 18:25:12,100 util INFO running with cStringIO, fabulous!
2013-09-16 18:25:12,122 main INFO Stetl version = 1.0.3
usage: stetl [-h] -c CONFIG_FILE [-s CONFIG_SECTION] [-a CONFIG_ARGS]
```

Especially check the Stetl version number.

Try running the examples when running with a downloaded distro.

```
cd examples/basics
./runall.sh
```

Look for any error messages in your output.

2.3 Run Unit Tests

You can run unit tests to completely verify your installation. First install some extra packages:

```
pip install -r requirements-dev.txt
```

Then run the tests using *nose2*.

```
nose2
```

2.4 Install with Docker

The fastest way to use Stetl is via [Docker](#). The Stetl Docker Image is lightweight, compressed just over 100MB, based on a Debian “slim” Docker Image.

Your environment needs to be setup to use Docker and probably you want to use some tooling like [Vagrant](#). The author uses a combination of VirtualBox with Ubuntu and Vagrant on Mac OSX to run Docker, but this is a bit out of scope here.

Assuming you have a working Docker environment, there are two ways to install Stetl with Docker:

- build a Docker image yourself using the Dockerfile in <https://github.com/geopython/stetl/blob/master/Dockerfile>
- use a prebuilt public Stetl Docker image from Docker Hub: <https://hub.docker.com/r/geopython/stetl>

When rebuilding you can add build arguments for your environment, defaults:

```
ARG TIMEZONE="Europe/Amsterdam"
ARG LOCALE="en_US.UTF-8"
ARG ADD_PYTHON_DEB_PACKAGES=""
ARG ADD_PYTHON_PIP_PACKAGES=""
```

For example building with extra Python packages:

```
docker build --build-arg ADD_PYTHON_DEB_PACKAGES="python-requests python-tz" -t_
↪geopython/stetl:latest .
docker build --build-arg ADD_PYTHON_PIP_PACKAGES="scikit-learn==0.18 influxdb" -t_
↪geopython/stetl:latest .
```

Or you may extend the Stetl Dockerfile with your own Dockerfile.

For running Stetl using Docker see [Using Docker](#).

The text below gives some introduction to ETL, the rationale why Stetl was developed and where and how it attempts to fit in.

3.1 Problem

Data conversion combined with model and coordinate transformation from a source to a target datastore (files, databases) is a recurring task in almost every geospatial project. This process is often referred to as **ETL (Extract Transform Load)**. Source and/or target geo-data formats are increasingly encoded as **GML (Geography Markup Language)**, either as flat records, so called Simple Features, but more and more using domain-specific, object oriented **OGC/ISO GML Application Schema's**.

GML Application Schema's are for example heavily used within the **INSPIRE Data Harmonization** effort in Europe. Many National Mapping and Cadastral Agencies (NMCAs) use GML-encoded datasets as their bulk format for download and exchange and via Web Feature Services (WFSs). As geospatial professionals we are often confronted with ETL-tasks involving (complex) GML or worse: "GML-lookalikes", which are often XML Schemas embedded with GML-namespaced elements.

Luckily, in many cases **GDAL/OGR**, the Swiss Army Knife for geo-data conversion, can do the job. If "ogr2ogr" sounds like gibberish to you, check out <http://gdal.org> ! But when complex, some say rich, GML Application Schemas are involved, data conversion can be a daunting task when GDAL/OGR alone is not sufficient. Firstly, often complex data model transformations have to be applied.

In addition we may be confronted with the bulkiness of GML:

- Megabyte/Gigabyte-files.
- Deeply nested elements where the nuggets, the actual attribute values, reside.
- Trees of .zip files and possibly more nasty surprises once we have unboxed a GML-delivery.
- High resource consumption in memory and CPU and long processing hours, up to complete machine-lockup, can be the side-effects of naive GML-processing.

3.2 Existing (partial) solutions

Within the FOSS4G world we can resort to high level, GUI-based, ETL-tools such as GeoKettle, Humboldt tools and Talend GeoSpatial. These are very powerful tools by themselves, check them out as well. Some of us, like the author, like to stay closer to GDAL/OGR and XSLT for model transforms, some command line tools and a bit of Python scripting, but without having to write a complete, ad-hoc ETL-program each time. This is the space where Stetl tries to fit in, so read on.

We already have great FOSS tools for XML/GML parsing, data-conversion and model-transformation like GDAL/OGR (`ogr2ogr!`), XSLT (Extensible Stylesheet Language Transformations, for transforming XML) and native XML-parsing libraries like `libxml2`. Each individual tool/library is extremely powerful and performant by itself. But we would like to combine of these tools. Take for example flat, national adres data in a PostGIS database that we need to transform to multiple INSPIRE Application Schema GML files. Each individual FOSS tool can handle part of the ETL: `ogr2ogr` for converting from PostGIS (including coordinate tranformation) into to simple feature GML, XSLT (`xsltproc/libxslt`) to transform the resulting flat GML to rich INSPIRE GML. But with millions of addresses we cannot simply use a single GML memory datastructure (DOM) or single intermediate GML-file.

3.3 Stetl: Python, streaming and configuration

Add Python and a configuration convention to this equation and we have Stetl: Streaming ETL. Stetl is a lightweight, geospatial ETL (Extract Transform Load) framework written in Python. ETL-processing with Stetl is driven from a configuration file. Within a Stetl configuration file a chain of ETL-processing modules is declared through which the data flows (“streams”). A module may be an input, filter or output module. Modules have input and output data types declared such that only compatible modules can be connected. However, Stetl does not define a grand internal data structure to which all data is mapped as many ETL-tools do. Data formats are kept close to the external tools that Stetl uses.

Stetl comes with pre-defined modules for:

- GML-parsing
- XSLT processing
- XSD Validation
- PostGIS/OGR input and output
- GML-splitting
- ... and many more.

Stetl calls on the above tools like OGR, `libxslt` and `libxml2` via their native interfaces. Stetl is even more speed-optimized as no intermediate file-storage is used: we use other means such as native string buffers. For example large XML/GML-files can be split into manageable documents and streamed into an XSLT filter module. Stetl-modules are off course extensible and can be user-defined. Reusable ETL-configurations invoked through parameterized commandline scripts can be defined without programming.

Stetl evolved from and is used within the INSPIRE-FOSS project (<http://inspire-foss.org>). Here for example, Dutch national addresses (BAG) were transformed into INSPIRE Addresses GML (files and database). Special Stetl integration modules are available to extract and publish data from/to a deegree WFS and deegree “Blobstore-database”. The combination Stetl/deegree is an ideal setup for INSPIRE deployments.

Other Dutch national datasets like Top10NL and BGT (Dutch topo vector datasets) have been completely and successfully transformed. Work is in progress to use Stetl as the basis for NLEExtract (<http://nlextract.nl>), a project that provides ETL tools for Dutch open geo-datasets. Stetl development is now (april 2013) in an initial phase and takes place in GitHub. The current version is workable but we hope to present a v1.0 at FOSS4G with more documentation

and as a standard Python Package via PyPi. The main link is: <http://stetl.org> (now links to GitHub). To get started find some basic examples here: <https://github.com/geopython/stetl/tree/master/examples/basics>.

This section explains how to use Stetl for your ETL. It assumes Stetl is installed and you are able to run the examples. It may be useful to study some of the examples, especially the core ones found in the `examples/basics` directory. These examples start numbering from 1, building up more complex ETL cases like [\(INSPIRE\) transformation using Jinja2 Templating](#).

In addition there are example cases like the Dutch Topo map (Top10NL) ETL in the `examples/top10nl` directory .

The core concepts of Stetl remain pretty simple: an input resource like a file or a database table is mapped to an output resource (also a file, a database, a remote HTTP server etc) via one or more filters. The input, filters and output are connected in a pipeline called a *processing chain* or Chain. This is a bit similar to a current in electrical engineering: an input flows through several filters, that each modify the current. In our case the current is (geospatial) data. Stetl design follows the so-called [Pipes and Filters Architectural Pattern](#).

4.1 Stetl Config

Stetl components (Inputs, Filters, Outputs) and their interconnection (the Pipeline/Chain) are specified in a Stetl config file. The file format follows the Python `.ini` file-format.

To illustrate, let's look at the example `2_xslt`. This example takes the input file `input/cities.xml` and transforms this file to a valid GML file called `output/gmlcities.gml`. The Stetl config file looks as follows.

```
[etl]
chains = input_xml_file|transformer_xslt|output_file

[input_xml_file]
class = inputs.fileinput.XmlFileInput
file_path = input/cities.xml

[transformer_xslt]
class = filters.xsltfilter.XsltFilter
script = cities2gml.xsl
```

(continues on next page)

(continued from previous page)

```
[output_file]
class = outputs.fileoutput.FileOutput
file_path = output/gmlcities.gml
```

Most of the sections in this ini-file specify a Stetl component: an Input, Filter or Output component. Each component is specified by its (Python) class and per-component specific parameters. For example [input_xml_file] uses the class `inputs.fileinput.XmlFileInput` reading and parsing the file `input/cities.xml` specified by the `file_path` property. [transformer_xslt] is a Filter that applies XSLT with the script file `cities2gml.xsl` that is in the same directory. The [output_file] component specifies the output, in this case a file.

These components are coupled in a Stetl *Chain* using the special .ini section [etl]. That section specifies one or more processing chains. Each Chain is specified by the names of the component sections, their interconnection using a the Unix pipe symbol “|”.

So the above Chain is `input_xml_file|transformer_xslt|output_file`. The names of the component sections like [input_xml_file] are arbitrary.

Note: since v1.1.0 a datastream can be split (see below) to multiple Outputs using () like :

```
[etl]
chains = input_xml_file|transformer_xslt|(output_gml_file)(output_wfs)
```

Or multiple Input streams can be combined/merged like:

```
[etl]
chains = (input_http_api_1) (input_http_api_2) | data_transformer | output_db
```

It is even possible to have both Splitting and Merging together with filtering:

```
[etl]
chains = (input_http_api_1 | cleaner_filter) (input_http_api_2) | data_transformer |
↪(output_db) (output_file)
```

4.2 Configuring Components

Most Stetl Components, i.e. inputs, filters, outputs, have properties that can be configured within their respective [section] in the config file. But what are the possible properties, values and defaults? This is documented within each Component class using the @Config decorator much similar to the standard Python @property, only with some more intelligence for type conversions, defaults, required presence and documentation. It is loosely based on https://wiki.python.org/moin/PythonDecoratorLibrary#Cached_Properties and Bruce Eckel’s <http://www.artima.com/weblogs/viewpost.jsp?thread=240845> with a fix/hack for Sphinx documentation.

See for example the `stetl.inputs.fileinput.FileInput` documentation.

For class authors: this information is added via the Python Decorators much similar to @property. The `stetl.component.Config` is used to define read-only properties for each Component instance. For example,

```
class FileInput (Input):
    """
    Abstract base class for specific FileInputs, use derived classes.
    """

    # Start attribute config meta
    # Applying Decorator pattern with the Config class to provide
```

(continues on next page)

(continued from previous page)

```

# read-only config values from the configured properties.

@Config(ptype=str, default=None, required=False)
def file_path(self):
    """
    Path to file or files or URLs: can be a dir or files or URLs
    or even multiple, comma separated. For URLs only JSON is supported now.
    """
    pass

@Config(ptype=str, default='*.[gxGX][mM][lL]', required=False)
def filename_pattern(self):
    """
    Filename pattern according to Python ``glob.glob`` for example:
    '\\*.[gxGX][mM][lL]'
    """
    pass

@Config(ptype=bool, default=False, required=False)
def depth_search(self):
    """
    Should we recurse into sub-directories to find files?
    """
    pass

# End attribute config meta

def __init__(self, configdict, section, produces):
    Input.__init__(self, configdict, section, produces)

    # Create the list of files to be used as input
    self.file_list = Util.make_file_list(self.file_path, None, self.filename_
↪pattern, self.depth_search)

```

This defines three configurable properties for the class `FileInput`. Each `@Config` has three parameters: `ptype`, the Python type (`str`, `list`, `dict`, `bool`, `int`), `default` (default value if not present) and `required` (if property is mandatory or optional).

Within the config one can set specific config values like,

```

[input_xml_file]
class = inputs.fileinput.XmlFileInput
file_path = input/cities.xml

```

This automatically assigns `file_path` to `self.file_path` without any custom code and assigns the default value to `filename_pattern`. Automatic checks are performed: if `file_path` (`required=True`) is present, if its type is string. In some cases type conversions may be applied e.g. when type is `dict` or `list`. It is guarded that the value is not overwritten and the docstrings will appear in the auto-generated documentation, each entry prepended with a `CONFIG` tag.

4.3 Running Stetl

The above ETL spec can be found in the file `etl.cfg`. Now Stetl can be run, simply by typing

```
stetl -c etl.cfg
```

Stetl will parse `etl.cfg`, create all Components by their class name and link them in a Chain and execute that Chain. Of course this example is very trivial, as we could just call XSLT without Stetl. But it becomes interesting with more complex transformations.

Suppose we want to convert the resulting GML to an *ESRI Shapefile*. As we cannot use GDAL `ogr2ogr` on the input file, we need to combine XSLT and `ogr2ogr`. See example [3_shape](#). Now we replace the output by using `outputs.ogroutput.Ogr2OgrOutput`, which can execute any `ogr2ogr` command, converting whatever it gets as input from the previous Filter in the Chain.

```
[etl]
chains = input_xml_file|transformer_xslt|output_ogr_shape

[input_xml_file]
class = inputs.fileinput.XmlFileInput
file_path = input/cities.xml

[transformer_xslt]
class = filters.xsltfilter.XsltFilter
script = cities2gml.xsl

# The ogr2ogr command-line. May be split over multiple lines for readability.
# Backslashes not required in that case.
[output_ogr_shape]
class = outputs.ogroutput.Ogr2OgrOutput
temp_file = temp/gmlcities.gml
ogr2ogr_cmd = ogr2ogr
               -overwrite
               -f "ESRI Shapefile"
               -a_srs epsg:4326
               output/gmlcities.shp
               temp/gmlcities.gml
```

4.3.1 Using Docker

The most convenient way to run Stetl is via Docker. See the installation instructions at *Install with Docker*. A full example can be viewed in the Smart Emission project: <https://github.com/Geonovum/smartemission/tree/master/etl>.

In the simplest case you run a Stetl Docker container from your own built image or the Dockerhub-provided one, `geopython/stetl:<version>` stetl as follows (*latest* version):

```
sudo docker run -v <host dir>:<container dir> -w <work dir> geopython/stetl:latest_
↳stetl <any Stetl arguments>
```

For example within the current directory you may have an `etl.cfg` Stetl file:

```
WORK_DIR=`pwd`
sudo docker run -v ${WORK_DIR}:${WORK_DIR} -w ${WORK_DIR} geopython/stetl:latest_
↳stetl -c etl.cfg
```

A more advanced setup would be (network-)linking to a PostGIS Docker image like `kartoza/postgis`:

```
# First run Postgis, remains running,
sudo docker run --name postgis -d -t kartoza/postgis:9.4-2.1
```

(continues on next page)

(continued from previous page)

```
# Then later run Stetl
STETL_ARGS="-c etl.cfg -a local.args"
WORK_DIR="`pwd`"

sudo docker run --name stetl --link postgis:postgis -v ${WORK_DIR}:${WORK_DIR} -w $
↪ ${WORK_DIR} geopython/stetl:latest stetl ${STETL_ARGS}
```

The last example is used within the SmartEmission project. Also with more detail and keeping all dynamic data (like PostGIS DB), your Stetl config and results, and logs within the host. For PostGIS see: <https://github.com/Geonovum/smartemission/tree/master/services/postgis> and Stetl see: <https://github.com/Geonovum/smartemission/tree/master/etl>.

Even better is to use *docker-compose*.

4.4 Stetl Integration

Note: one can also run Stetl via its main ETL class: `stetl.etl.ETL`. This may be useful for integrations in for example Python programs or even OGC WPS servers (planned).

4.5 Reusable Stetl Configs

What we saw in the last example is that it is hard to reuse this *etl.cfg* when we have for example a different input file or want to map to different output files. For this Stetl supports *config parameter substitution*.

Dynamic or secret (e.g. database credentials) parameters in *etl.cfg* are declared symbolically and substituted at runtime via the commandline or the OS environment (new in 2018).

A variable is declared between curly brackets like `{out_xml}`. See example [6_cmdargs](#).

```
[etl]
chains = input_xml_file|transformer_xslt|output_file

[input_xml_file]
class = inputs.fileinput.XmlFileInput
file_path = {in_xml}

[transformer_xslt]
class = filters.xsltfilter.XsltFilter
script = {in_xslt}

[output_file]
class = outputs.fileoutput.FileOutput
file_path = {out_xml}
```

Note the symbolic input, xsl and output files. We can now perform this ETL using the *stetl -a option* in two ways. One, passing the arguments on the commandline, like

```
stetl -c etl.cfg -a "in_xml=input/cities.xml in_xslt=cities2gml.xslt out_xml=output/
↪ gmlcities.gml"
```

Two, passing the arguments in a properties file, here called *etl.args* (the name of the suffix `.args` is not significant, could be `.env` as well).

```
stetl -c etl.cfg -a etl.args
```

Where the content of the *etl.args* properties file is:

```
# Arguments in properties file
in_xml=input/cities.xml
in_xsl=cities2gml.xsl
out_xml=output/gmlcities.gml
```

A third way is to pass these key/value pairs (partly) as OS Environment variables. This is especially handy in Docker-based deployments like Docker Compose and Kubernetes. In this case the variable names need to be prepended with *STETL_* or *stetl_* as to not mix-up with other non-related OS-env vars. A mixture of commandline args (file) and environment vars is possible. The rule is that *OS Environment variables always take prevalence*.

For example, the above args could also be passed as follows:

```
export stetl_in_xml="input/cities.xml"
export stetl_in_xsl="cities2gml.xsl"
export stetl_out_xml="output/gmlcities.gml"
stetl -c etl.cfg
```

or only override the input file name from *etl.args*:

```
export stetl_in_xml="input/cities2.xml"
stetl -c etl.cfg -a etl.args
```

This makes an ETL chain highly reusable. A very elaborate Stetl config with parameter substitution can be seen in the [Top10NL ETL](#).

4.6 Connection Compatibility

During ETL Chain processing Components typically pass data to a next *stetl.component.Component*. A *stetl.filter.Filter* Component both consumes and produces data, Inputs produce data and Outputs only consume data.

Data and status flows as *stetl.packet.Packet* objects between the Components. The type of the data in these Packets needs to be compatible only between two coupled Components. Stetl does not define one unifying data structure, but leaves this to the Components themselves.

Each Component provides the type of data it *consumes* (Filters, Outputs) and/or *produces* (Inputs, Filters). This is indicated in its class definition using the *consumes* and *produces* object constructor parameters. Some Components can produce and/or consume multiple data types, like a single stream of *records* or a *record array*. In those cases the *produces* or *consumes* parameter can be a list (array) of data types.

During *Chain* construction Stetl will check for compatible formats when connecting *Components*. If one of the formats is a list of formats, the actual format is determined by:

1. explicit setting: the actual *input_format* and/or *output_format* is set in the Component *.ini* configuration
2. no setting provided: the first format in the list is taken as default

Stetl will only check if these input and output-formats for connecting Components are compatible when constructing a Chain.

The following data types are currently symbolically defined in the *stetl.packet.FORMAT* class:

- any - 'catch-all' type, may be any of the types below.

- `etree_doc` - a complete in-memory XML DOM structure using the lxml etree
- `etree_element` - each Packet contains a single DOM Element (usually a Feature) in lxml etree format
- `etree_feature_array` - each Packet contains an array of DOM Elements (usually Features) in lxml etree format
- `geojson_feature` - as struct but following naming conventions for a single Feature according to the GeoJSON spec: <http://geojson.org>
- `geojson_collection` - as struct but following naming conventions for a FeatureCollection according to the GeoJSON spec: <http://geojson.org>
- `ogr_feature` - a single Feature object from an OGR source (via Python SWIG wrapper)
- `ogr_feature_array` - a Python list (array) of a single Feature objects from an OGR source
- `record` - a Python dict (hashmap)
- `record_array` - a Python list (array) of dict
- `string` - a general string
- `struct` - a JSON-like generic tree structure
- `xml_doc_as_string` - a string representation of a complete XML document
- `xml_line_stream` - each Packet contains a line (string) from an XML file or string representation (DEPRECATED)

Many components, in particular Filters, are able to transform data formats. For example the *XmlElementStreamer-FileInput* can produce an *etree_element*, a subsequent *XmlAssembler* can create small in-memory *etree_doc* s that can be fed into an *XsltFilter*, which outputs a transformed *etree_doc*. The type *any* is a catch-all, for example used for printing any object to standard output in the `stetl.packet.Component`. An *etree_element* may also be interesting to be able to process single features.

Starting with Stetl 1.0.7 a new `stetl.filters.formatconverter.FormatConverterFilter` class provides a Stetl Filter to allow almost any conversion between otherwise incompatible Components.

TODO: the Packet typing system is still under constant review and extension. Soon it will be possible to add new data types and converters. We have deliberately chosen not to define a single internal datatype like a “Feature”, both for flexibility and performance reasons.

4.7 Multiple Chains

Usually a complete ETL will require multiple steps/commands. For example we need to create a database, maybe tables and/or making tables empty. Also we may need to do postprocessing, like removing duplicates in a table etc. In order to have repeatable/reusable ETL without any manual steps, we can specify multiple Chains within a single Stetl config. The syntax: chains are separated by commas (steps are sill separated by pipe symbols).

Chains are executed in order. We can even reuse the specified components from within the same file. Each will have a separate instance within a Chain.

For example in the [Top10NL example](#) we see three Chains:

```
[etl]
chains = input_sql_pre|schema_name_filter|output_postgres,
         input_big_gml_files|xml_assembler|transformer_xslt|output_ogr2ogr,
         input_sql_post|schema_name_filter|output_postgres
```

Here the Chain `input_sql_pre|schema_name_filter|output_postgres` sets up a PostgreSQL schema and creates tables. `input_big_gml_files|xml_assembler|transformer_xslt|output_ogr2ogr` does the actual ETL and `input_sql_post|schema_name_filter|output_postgres` does some PostgreSQL postprocessing.

4.8 Chain Splitting

In some cases we may want to split processed data to multiple `Filters` or `Outputs`. For example to produce output files in multiple formats like GML, GeoJSON etc or to publish converted (Filtered) data to multiple remote services (SOS, SensorThings API) or just for simple debugging to a target `Output` and `StandardOutput`.

See issue <https://github.com/geopython/stetl/issues/35> and the [Chain Split example](#).

Here the Chains are split by using `()` in the ETL Chain definition:

```
# Transform input xml to valid GML file using an XSLT filter and pass to multiple_
↳outputs.
# Below are two Chains: simple Output splitting and splitting to 3 sub-Chains at_
↳Filter level.

[etl]
chains = input_xml_file | transformer_xslt | (output_file) (output_std),
        input_xml_file | (transformer_xslt|output_file) (output_std) (transformer_
↳xslt|output_std)

[input_xml_file]
class = inputs.fileinput.XmlFileInput
file_path = input/cities.xml

[transformer_xslt]
class = filters.xsltfilter.XsltFilter
script = cities2gml.xsl

[output_file]
class = outputs.fileoutput.FileOutput
file_path = output/gmlcities.gml

[output_std]
class = outputs.standardoutput.StandardOutput
```

4.9 Chain Merging

In some cases we may want to merge (combine, join) multiple input streams.

For example to harvest data from multiple HTTP REST APIs, or to realize a *Filter* that integrates data from two data-sources.

See issue <https://github.com/geopython/stetl/issues/59> and the [Chain Merge example](#).

Here the Chains are merged by using `()` notation in the ETL Chain definition, possibly even combined with Splitting `Outputs`:

```
# Merge two inputs into single Filter.
```

(continues on next page)

(continued from previous page)

```
[etl]
chains = (input_1) (input_2)|transformer_xslt|output_std,
          (input_1) (input_2)|transformer_xslt|(output_file) (output_std)

[input_1]
class = inputs.fileinput.XmlFileInput
file_path = input1/cities.xml

[input_2]
class = inputs.fileinput.XmlFileInput
file_path = input2/cities.xml

[transformer_xslt]
class = filters.xsltfilter.XsltFilter
script = cities2gml.xsl

[output_file]
class = outputs.fileoutput.FileOutput
file_path = output/gmlcities.gml

[output_std]
class = outputs.standardoutput.StandardOutput
```


This chapter lists various cases/projects where Stetl is used.

5.1 NLExtract

NLExtract <http://nlextract.nl> is a development project that aims to provide ETL-tooling for all Dutch Open Geo-Datasets, in particular the country wide “Key Registries” (Dutch: Basisregistraties) like Cadastral Parcels (BRK), Topography (BRT+BGT) and Buildings and Addresses (BAG). These datasets are provided as XML/GML. The ETL mostly provides a transformation to PostGIS. For all Key Registries, except for the BAG, Stetl is used, basically as-is, without extra (Python) programming. See also the NLExtract GitHub: <https://github.com/nlextract/NLExtract>

5.1.1 Topography (BRT/Top10NL)

See <https://github.com/nlextract/NLExtract/tree/master/top10nl/etl> and the Stetl conf at <https://github.com/nlextract/NLExtract/tree/master/top10nl/etl/conf/>

5.1.2 Detailed Topography (BGT)

See <https://github.com/nlextract/NLExtract/tree/master/bgt> and the Stetl conf at <https://github.com/nlextract/NLExtract/blob/master/bgt/etl/conf/>

5.1.3 Cadastral Parcels (BRK)

See <https://github.com/nlextract/NLExtract/tree/master/brk/etl> and the Stetl conf at <https://github.com/nlextract/NLExtract/tree/master/brk/etl/conf/>

5.2 INSPIRE

These were the origins of Stetl. This project was sponsored by Kadaster. See <https://github.com/justb4/inspire-foss>. The ETL involved the transformation of Dutch Key Registries (see above) to harmonized INSPIRE GML according to the Annexes.

5.2.1 Addresses

BAG to INSPIRE Addresses Annex II Theme.

See <https://github.com/justb4/inspire-foss/blob/master/etl/NL.Kadaster/Addresses/>

5.3 Ordnance Survey

A successful Proof-of-Concept to convert Ordnance Survey Mastermap GML to PostGIS:

<https://github.com/geopython/stetl/tree/master/examples/ordnancesurvey>

5.4 SOSpilot

A SensorWeb project by Geonovum, see <http://sensors.geonovum.nl>.

5.4.1 Dutch AQ to WFS/WMS(-Time) and SOS

Stetl was used for ETL from Dutch Air Quality Data from RIVM (XML) to WMS(-Time), WFS and SOS. The latter was effected by SOS-Transactional publication. Documentation at <http://sospilot.readthedocs.org> and ETL on GitHub at <https://github.com/Geonovum/sospilot/tree/master/src/rivm-lml>

5.4.2 Dutch AQ to EAI Reporting

Stetl was used to generate XML-based reports for the EU EAI:

<https://github.com/Geonovum/sospilot/tree/master/src/aq-report>

This involved the first use of Jinja2 templating for complex XML/GML generation.

5.5 Smart Emission

Sensors for air quality, meteo and audio at civilians. Project by University of Nijmegen/Gemeente Nijmegen with participation by Geonovum. Stetl is used to transform a low-level sensor API to PostGIS and later on WMS/WFS/SOS and the SensorThings API. Also InfluxDB output is developed here.

This is also an example how to use a Stetl Docker image:

See <https://github.com/Geonovum/smartemission/tree/master/etl>

Below is the API documentation for the the Stetl Python code.

6.1 Main Entry Points

There are several entry points through which Stetl can be called. The most common is to use the commandline script *bin/stetl*. This command should be available after doing an install.

In some contexts like integrations you may want to call Stetl via Python. The entries are then.

`stetl.main.main()`

The *main* function, to be called from commandline, like *python src/main.py -c etl.cfg*.

Args:

- c** **--config <config_file>** the Stetl config file.
- s** **--section <section_name>** the section in the Stetl config (ini) file to execute (default is [etl]).
- a** **--args <arglist>** substitutable args for symbolic, {arg}, values in Stetl config file, in format "arg1=foo arg2=bar" etc.
- d** **--doc <class>** Get component documentation like its configuration parameters, e.g. `stetl -doc stetl.inputs.fileinput.FileInput`
- h** **--help** get help info

`stetl.main.print_doc(class_name)`

Print documentation for class in particular config options

class `stetl.etl.ETL(options_dict, args_dict=None)`

The main class: builds ETL Chains with connected Components from a config and let them run.

Usually this class is called via `main` but it may be called directly for direct integration.

env_expand_args_dict (*args_dict*, *args_names*)

Expand values in dict with equivalent values from the OS Env. NB vars in OS Env should be prefixed with *STETL_* or *stetl_* as to get overrides by accident.

Returns expanded *args_dict* or None

6.2 Core Framework

The core framework is directly under the directory *src/stetl*. Below are the main seven classes. Their interrelation is as follows:

One or more *stetl.chain.Chain* objects are built from a Stetl ETL configuration via the *stetl.factory.Factory* class. A *stetl.chain.Chain* consists of a set of connected *stetl.component.Component* objects. A *stetl.component.Component* is either an *stetl.input.Input*, an *stetl.output.Output* or a *stetl.filter.Filter*. Data and status flows as *stetl.packet.Packet* objects from an *stetl.input.Input* via zero or more *stetl.filter.Filter* objects to a final *stetl.output.Output*.

As a trivial example: an *stetl.input.Input* could be an XML file, a *stetl.filter.Filter* could represent an XSLT file and an *stetl.output.Output* a PostGIS database. This is effected by specialized classes in the subpackages *inputs*, *filters*, and *outputs*. New in 1.1.0: *stetl.Splitter* to split data to multiple *Outputs* and *stetl.Merger* to combine multiple *Inputs*.

class *stetl.factory.Factory*

Object and class Factory (Pattern). Based on: <http://stackoverflow.com/questions/2226330/instantiate-a-python-class-from-a-name>

class_forname (*class_string*)

Returns class instance specified by a string.

Args: *class_string*: The string representing a class.

Raises: ValueError if module part of the class is not specified.

new_instance (*class_obj*, *configdict*, *section*)

Returns object instance from class instance.

Args: *class_obj*: object representing a class instance. *args*: standard args. *kwargs*: standard args.

class *stetl.component.Component* (*configdict*, *section*, *consumes='none'*, *produces='none'*)

Abstract Base class for all Input, Filter and Output Components.

after_chain_invoke (*packet*)

Called right after entire Component Chain invoke.

after_invoke (*packet*)

Called right after Component invoke.

before_invoke (*packet*)

Called just before Component invoke.

exit ()

Allows derived Components to perform a one-time exit/cleanup.

init ()

Allows derived Components to perform a one-time init.

input_format ()

CONFIG The specific input format if the consumes parameter is a list or the format to be converted to the *output_format*.

- type: str
- required: False
- default: None

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

output_format ()

CONFIG The specific output format if the produces parameter is a list or the format to which the input format is converted.

- type: str
- required: False
- default: None

class `stetl.component.Config` (*ptype=<type 'str'>, default=None, required=False*)

Decorator class to tie config values from the .ini file to object instance property values. Somewhat like the Python standard @property but with the possibility to define default values, typing and making properties required.

Each property is defined by @Config(type, default, required). Basic idea comes from: https://wiki.python.org/moin/PythonDecoratorLibrary#Cached_Properties

class `stetl.chain.Chain` (*chain_str, config_dict*)

Holder for single invocable pipeline of components A Chain is basically a singly linked list of Components Each Component executes a part of the total ETL. Data along the Chain is passed within a Packet object. The compatibility of input and output for linked Components is checked when adding a Component to the Chain.

add (*etl_comp*)

Add component to end of Chain :param etl_comp: :return:

assemble ()

Builder method: build a Chain of linked Components :return:

get_by_class (*clazz*)

Get Component instance from Chain by class, mainly for testing. :param clazz: :return Component:

get_by_id (*id*)

Get Component instance from Chain, mainly for testing. :param name: :return Component:

get_by_index (*index*)

Get Component instance from Chain by position/index in Chain, mainly for testing. :param clazz: :return Component:

run ()

Run the ETL Chain. :return:

class `stetl.packet.FORMAT`

Format of Packet (enumeration).

Current possible values:

- 'none'
- 'xml_line_stream'
- 'line_stream'
- 'etree_doc'

- 'etree_element'
- 'etree_feature_array'
- 'xml_doc_as_string'
- 'string'
- 'record'
- 'record_array'
- 'struct'
- 'geojson_feature'
- 'geojson_collection'
- 'ogr_feature'
- 'ogr_feature_array'
- 'any'

class `stetl.packet.Packet` (*data=None*)
Represents units of (any) data and status passed along Chain of Components.

class `stetl.input.Input` (*configdict, section, produces*)
Bases: `stetl.component.Component`
Abstract Base class for all Input Components.

invoke (*packet*)
Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.output.Output` (*configdict, section, consumes*)
Bases: `stetl.component.Component`
Abstract Base class for all Output Components.

invoke (*packet*)
Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.filter.Filter` (*configdict, section, consumes, produces*)
Bases: `stetl.component.Component`
Maps input to output. Abstract base class for specific Filters.

invoke (*packet*)
Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.splitter.Splitter` (*config_dict, child_list*)
Bases: `stetl.component.Component`

Component that splits a single input to multiple output Components. Use this for example to produce multiple output file formats (GML, GeoJSON etc) or to publish to multiple remote services (SOS, SensorThings API) or for simple debugging: target Output and StandardOutput.

after_chain_invoke (*packet*)
Called right after entire Component Chain invoke.

after_invoke (*packet*)
Called right after Component invoke.

before_invoke (*packet*)
Called just before Component invoke.

exit ()

Allows derived Components to perform a one-time exit/cleanup.

init ()

Allows derived Components to perform a one-time init.

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.merger.Merger` (*config_dict, child_list*)

Bases: `stetl.component.Component`

Component that merges multiple Input Components into a single Component. Use this for example to combine multiple input streams like API endpoints. The Merger will embed Child Components to which actions are delegated. A Child Component may be a sub-Chain e.g. (InputFilter|Filter..) sequence. Hence the “next” should be coupled to the last Component in that sub-Chain with the degenerate case where the sub-Chain is a single (Input) Component. NB this Component can only be used for Inputs.

first (*child*)

Get first Component in Child sub-Chain. :param child: :return: first Component

last (*child*)

Get last Component in Child sub-Chain. :param child: :return: last Component

6.3 Components: Inputs

class `stetl.inputs.dbinput.DbInput` (*configdict, section, produces*)

Bases: `stetl.input.Input`

Input from any database (abstract base class).

class `stetl.inputs.dbinput.PostgresDbInput` (*configdict, section*)

Bases: `stetl.inputs.dbinput.SqlDbInput`

Input by querying records from a Postgres database. Input is a query, like SELECT * from mytable. Output is zero or more records as record array (array of dict) or single record (dict).

produces=FORMAT.record_array (default) or FORMAT.record

exit ()

Allows derived Components to perform a one-time exit/cleanup.

host ()

CONFIG host name or host IP-address, defaults to ‘localhost’

- type: str
- required: False
- default: localhost

init ()

Allows derived Components to perform a one-time init.

password ()

CONFIG User password, defaults to ‘postgres’

- type: str

- required: False
- default: postgres

port ()

CONFIG port for host, defaults to '5432'

- type: str
- required: False
- default: 5432

raw_query (*query_str*)

Performs DB-specific query and returns raw records iterator.

schema ()

CONFIG The postgres schema name, defaults to 'public'

- type: str
- required: False
- default: public

user ()

CONFIG User name, defaults to 'postgres'

- type: str
- required: False
- default: postgres

class `stetl.inputs.dbinput.SqlDbInput` (*configdict, section*)

Bases: `stetl.inputs.dbinput.DbInput`

Input using a query from any SQL-based RDBMS (abstract base class).

column_names ()

CONFIG Column names to populate records with. If empty taken from table metadata.

- type: str
- required: False
- default: None

database_name ()

CONFIG Database name

- type: str
- required: True
- default: None

do_query (*query_str*)

DB-neutral query returning Python record list.

query ()

CONFIG The query (string) to fire.

- type: str
- required: False
- default: None

raw_query (*query_str*)

Performs DB-specific query and returns raw records iterator.

read_once ()

CONFIG Read once? i.e. only do query once and stop

- type: bool
- required: False
- default: False

result_to_output (*db_tuples*)

Convert DB-specific record tuples to single Python record (dict) or record array (list of dict).

table ()

CONFIG Table name

- type: str
- required: False
- default: None

tuples_to_records (*db_tuples, columns=None*)

Convert tuple array (list of tuple) to list of records (list of dict's) using list of column names.

class `stetl.inputs.dbinput.SQLiteDbInput` (*configdict, section*)

Bases: `stetl.inputs.dbinput.SqlDbInput`

Input by querying records from a SQLite database. Input is a query, like `SELECT * from mytable`. Output is zero or more records as record array (array of dict) or single record (dict).

produces=FORMAT.record_array (default) or FORMAT.record

init ()

Allows derived Components to perform a one-time init.

raw_query (*query_str*)

Performs DB-specific query and returns raw records iterator.

class `stetl.inputs.fileinput.ApacheLogFileInput` (*configdict, section*)

Bases: `stetl.inputs.fileinput.FileInput`

Parses Apache log files. Lines are converted into records based on the log format. Log format should follow Apache Log Format. See `ApacheLogParser` for details.

produces=FORMAT.record

key_map ()

CONFIG Map of cryptic %-field names to readable keys in record.

- type: dict
- required: False
- default: { '%l': 'logname', '%{Referer}i': 'referer', '%>s': 'status', '%t': 'time', '%u': 'user', '%D': 'deltat', '%r': 'request', "%%h": 'host', '%{User-agent}i': 'agent', '%b': 'bytes' }

log_format ()

CONFIG Log format according to Apache CLF

- type: str
- required: False
- default: %h %l %u %t "%r" %>s %b "%{Referer}i" "%{User-agent}i"

class `stetl.inputs.fileinput.CsvFileInput` (*configdict, section*)

Bases: `stetl.inputs.fileinput.FileInput`

Parse CSV file into stream of records (dict structures) or a one-time record array. NB raw version: CSV needs to have first line with fieldnames.

produces=FORMAT.record or FORMAT.record_array

delimiter ()

CONFIG A one-character string used to separate fields. It defaults to ','.

- type: str
- required: False
- default: ,

init ()

Allows derived Components to perform a one-time init.

quote_char ()

CONFIG A one-character string used to quote fields containing special characters, such as the delimiter or quotechar, or which contain new-line characters. It defaults to '"'.

- type: str
- required: False
- default: "

class `stetl.inputs.fileinput.FileInput` (*configdict, section, produces*)

Bases: `stetl.input.Input`

Abstract base class for specific FileInputs, use derived classes.

depth_search ()

CONFIG Should we recurse into sub-directories to find files?

- type: bool

- required: False
- default: False

file_path()

CONFIG Path to file or files or URLs: can be a dir or files or URLs or even multiple, comma separated.
For URLs only JSON is supported now.

- type: str
- required: False
- default: None

filename_pattern()

CONFIG Filename pattern according to Python `glob.glob` for example: `*.[gxGX][mM][IL]`

- type: str
- required: False
- default: `*.[gxGX][mM][IL]`

read_file(file_path)

Override in subclass.

class `stetl.inputs.fileinput.GlobFileInput` (*configdict*, *section*, *produces=['string', 'line_stream']*)

Bases: `stetl.inputs.fileinput.FileInput`

Returns file names based on the `glob.glob` pattern given as `filename_filter`.

`produces=FORMAT.string` or `FORMAT.line_stream`

class `stetl.inputs.fileinput.JsonFileInput` (*configdict*, *section*)

Bases: `stetl.inputs.fileinput.FileInput`

Parse JSON file from file system or URL into hierarchical data struct. The struct format may also be a GeoJSON structure. In that case the `output_format` needs to be explicitly set to `geojson_collection` in the component config.

`produces=FORMAT.struct` or `FORMAT.geojson_collection`

read_file(file_path)

Override in subclass.

class `stetl.inputs.fileinput.LineStreamerFileInput` (*configdict*, *section*, *produces='line_stream'*)

Bases: `stetl.inputs.fileinput.FileInput`

Reads text-files, producing a stream of lines, one line per Packet. NB assumed is that lines in the file have newlines !!

process_line(line)

Override in subclass.

class `stetl.inputs.fileinput.StringFileInput` (*configdict*, *section*)

Bases: `stetl.inputs.fileinput.FileInput`

Reads and produces file as String.

`produces=FORMAT.string`

format_args()

CONFIG Formatting of content according to Python String.format() Input file should have substitutable values like {schema} {foo} format_args should be of the form format_args = schema:test foo:bar

- type: str
- required: False
- default: None

read_file (*file_path*)
Overridden from base class.

class stetl.inputs.fileinput.**XmlElementStreamerFileInput** (*configdict, section*)
Bases: *stetl.inputs.fileinput.FileInput*

Extracts XML elements from a file, outputs each feature element in Packet. Parsing is streaming (no internal DOM buildup) so any file size can be handled. Use this class for your big GML files!

produces=FORMAT.etree_element

element_tags ()

CONFIG Comma-separated string of XML (feature) element tag names of the elements that should be extracted and added to the output element stream.

- type: list
- required: True
- default: None

strip_namespaces ()

CONFIG should namespaces be removed from the input document and thus not be present in the output element stream?

- type: bool
- required: False
- default: False

class stetl.inputs.fileinput.**XmlFileInput** (*configdict, section*)
Bases: *stetl.inputs.fileinput.FileInput*

Parses XML files into etree docs (do not use for large files!).

produces=FORMAT.etree_doc

read_file (*file_path*)
Override in subclass.

class stetl.inputs.fileinput.**XmlLineStreamerFileInput** (*configdict, section*)
Bases: *stetl.inputs.fileinput.LineStreamerFileInput*

DEPRECATED Streams lines from an XML file(s) NB assumed is that lines in the file have newlines !! DEPRECATED better is to use XmlElementStreamerFileInput for GML features.

produces=FORMAT.xml_line_stream

class `stetl.inputs.fileinput.ZipFileInput` (*configdict, section*)

Bases: `stetl.inputs.fileinput.FileInput`

Parse ZIP file from file system or URL into a stream of records containing zipfile-path and file names.

produces=FORMAT.record

name_filter ()

CONFIG Regular “glob.glob” expression for filtering out filenames from the ZIP archive.

- type: str
- required: False
- default: *

class `stetl.inputs.httpinput.ApacheDirInput` (*configdict, section, produces='record'*)

Bases: `stetl.inputs.httpinput.HttpInput`

Read file data from an Apache directory “index” HTML page. Uses <http://stackoverflow.com/questions/686147/url-tree-walker-in-python> produces=FORMAT.record. Each record contains file_name and file_data (other meta data like date time is too fragile over different Apache servers).

file_ext ()

CONFIG The file extension for target files in Apache dir.

- type: str
- required: False
- default: xml

filter_file (*file_name*)

Filter the file_name, e.g. to suppress reading, default: return file_name.

Parameters `file_name` –

Return string or None

init ()

Read the list of files from the Apache index URL.

next_file ()

Return a tuple (name, date, size) with next file info.

Return tuple

no_more_files ()

More files left?.

Return Boolean

read (*packet*)

Read the data from the URL.

Parameters `packet` –

Returns

class `stetl.inputs.httpinput.HttpInput` (*configdict, section, produces='any'*)

Bases: `stetl.input.Input`

Fetch data from remote services like WFS via HTTP protocol. Base class: subclasses will do datatype-specific formatting of the returned data.

produces=FORMAT.any

add_authorization (*request*)

Add authorization from config data. Authorization scheme-specific. May be extended or overloaded for additional schemes.

Parameters *request* – the HTTP Request

Returns

auth ()

CONFIG Authentication data: Flat JSON-like struct dependent on auth type/schema. Only the *type* field is required, other fields depend on auth schema. Supported values :

```
type: basic|token
```

If the type is *basic* (HTTP Basic Authentication) two additional fields *user* and *password* are required. If the type is *token* (HTTP Token) additional two additional fields *keyword* and *token* are required.

Any required Base64 encoding is provided by `HttpInput`.

Examples:

```
# Basic Auth
url = https://some.rest.api.com
auth = {
  type: basic,
  user: myname
  password: mypassword
}

# Token Auth
url = https://some.rest.api.com
auth = {
  type: token,
  keyword: Bearer
  token: mytoken
}
```

- type: dict
- required: False
- default: None

format_data (*data*)

Format response data, override in subclasses, defaults to returning original data. :param packet: :return:

parameters ()

CONFIG Flat JSON-like struct of the parameters to be appended to the url.

Example: (parameters require quotes):


```
url = http://geodata.nationaalgeoregister.nl/natura2000/wfs
parameters = {
    service : WFS,
    version : 1.1.0,
    request : GetFeature,
    srsName : EPSG:28992,
    outputFormat : text/xml; subtype=gml/2.1.2,
    typename : natura2000
}
```

- type: dict
- required: False
- default: None

read (*packet*)

Read the data from the URL.

Parameters *packet* –

Returns

read_from_url (*url*, *parameters=None*)

Read the data from the URL.

Parameters

- **url** – the url to fetch
- **parameters** – optional dict of query parameters

Returns

url ()

CONFIG The HTTP URL string.

- type: str
- required: True
- default: None

class `stetl.inputs.ogrinput.OgrInput` (*configdict*, *section*)

Bases: `stetl.input.Input`

Direct GDAL OGR input via Python OGR wrapper. Via the Python API <http://gdal.org/python> an OGR data source is accessed and from each layer the Features are read. Each Layer corresponds to a “doc”, so for multi-layer sources the ‘end-of-doc’ flag is set after a Layer has been read.

This input can read almost any geospatial dataformat. One can use the features directly in a Stetl Filter or use a converter to e.g. convert to GeoJSON structures.

produces=FORMAT.ogr_feature or FORMAT.ogr_feature_array (all features)

data_source ()

CONFIG String denoting the OGR datasource. Usually a path to a file like “path/rivers.shp” or connection string to PostgreSQL like “PG: host=localhost dbname='rivers' user='postgres’”.

- type: str

- required: True
- default: None

init ()

Allows derived Components to perform a one-time init.

source_format ()

CONFIG Instructs GDAL to use driver by that name to open datasource. Not required for many standard formats that are self-describing like ESRI Shapefile.

Examples: 'PostgreSQL', 'GeoJSON' etc

- type: str
- required: False
- default: None

source_options ()

CONFIG Custom datasource-specific options. Used in `gdal.SetConfigOption()`.

- type: dict
- required: False
- default: None

sql ()

CONFIG String with SQL query. Mandatory for PostgreSQL OGR source.

- type: str
- required: False
- default: None

class `stetl.inputs.ogrinput.OgrPostgisInput` (*configdict, section*)

Bases: `stetl.input.Input`

Input from PostGIS via `ogr2ogr` command. For now hardcoded to produce an ogr GML line stream. `OgrInput` may be a better alternative.

Alternatives: either `stetl.input.PostgresqlInput` or `stetl.input.OgrInput`.

produces=FORMAT.xml_line_stream

in_pg_db ()

CONFIG Database name input DB.

- type: str
- required: True
- default: None

in_pg_host ()

CONFIG Host of input DB.

- type: str
- required: False
- default: localhost

in_pg_password()

CONFIG Password input DB.

- type: str
- required: False
- default: postgres

in_pg_port()

CONFIG Port of input DB.

- type: str
- required: False
- default: 5432

in_pg_schema()

CONFIG DB Schema name input DB.

- type: str
- required: False
- default: None

in_pg_sql()

CONFIG The input query (string) to fire.

- type: str
- required: False
- default: None

in_pg_user()

CONFIG User input DB.

- type: str
- required: False
- default: postgres

in_srs()

CONFIG SRS (projection) (ogr2ogr -s_srs) input DB e.g. 'EPSG:28992'.

- type: str
- required: False

- default: None

init ()

Allows derived Components to perform a one-time init.

out_dimension ()

CONFIG Dimension (OGR: DIM=N) of features in output stream.

- type: str
- required: False
- default: 2

out_geotype ()

CONFIG OGR Geometry type new layer in output stream, e.g. POINT.

- type: str
- required: False
- default: None

out_gml_format ()

CONFIG GML format OGR name in output stream, e.g. 'GML3'.

- type: str
- required: False
- default: None

out_layer_name ()

CONFIG New Layer name (ogr2ogr -nlm) output stream, e.g. 'address'.

- type: str
- required: False
- default: None

out_srs ()

CONFIG Target SRS (ogr2ogr -t_srs) code output stream.

- type: str
- required: False
- default: None

class `stetl.inputs.deegreeinput.DeegreeBlobstoreInput` (*configdict, section*)

Bases: `stetl.input.Input`

Read features from deegree Blobstore DB into an etree doc.

produces=FORMAT.etree_doc

end_container ()

CONFIG Tag that ends container.

- type: str
- required: True
- default: None

end_feature_tag()

CONFIG XML tag that ends Feature.

- type: str
- required: False
- default: None

init()

Allows derived Components to perform a one-time init.

max_features_per_doc()

CONFIG Max features to read from input feature GML stream per internal document.

- type: int
- required: False
- default: 10000

start_container()

CONFIG Tag that starts container.

- type: str
- required: True
- default: None

start_feature_tag()

CONFIG XML tag that starts Feature.

- type: str
- required: False
- default: False

6.4 Components: Filters

class `stetl.filters.xsltfilter.XsltFilter` (*configdict, section*)

Bases: `stetl.filter.Filter`

Invokes XSLT processor (via lxml) for given XSLT script on an etree doc.

consumes=FORMAT.etree_doc, produces=FORMAT.etree_doc

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.filters.xmlassembler.XmlAssembler` (*configdict, section*)

Bases: `stetl.filter.Filter`

Split a stream of etree DOM XML elements (usually Features) into etree DOM docs. Consumes and buffers elements until `max_elements` reached, will then produce an etree doc.

`consumes=FORMAT.etree_element, produces=FORMAT.etree_doc`

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.filters.xmlelementreader.XmlElementReader` (*configdict, section*)

Bases: `stetl.filter.Filter`

Extracts XML elements from a file, outputs each feature element in Packet. Parsing is streaming (no internal DOM buildup) so any file size can be handled. Use this class for your big GML files!

`consumes=FORMAT.string, produces=FORMAT.etree_element`

element_tags ()

CONFIG Comma-separated string of XML (feature) element tag names of the elements that should be extracted and added to the output element stream.

- type: list
- required: True
- default: None

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

strip_namespaces ()

CONFIG should namespaces be removed from the input document and thus not be present in the output element stream?

- type: bool
- required: False
- default: False

class `stetl.filters.xmlvalidator.XmlSchemaValidator` (*configdict, section*)

Bases: `stetl.filter.Filter`

Validates an etree doc and prints result to log.

`consumes=FORMAT.etree_doc, produces=FORMAT.etree_doc`

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.filters.stringfilter.StringFilter` (*configdict, section, consumes, produces*)

Bases: `stetl.filter.Filter`

Base class for any string filtering

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.filters.stringfilter.StringSubstitutionFilter` (*configdict, section*)

Bases: `stetl.filters.stringfilter.StringFilter`

String filtering using Python advanced String formatting. String should have substitutable values like {schema} {foo} `format_args` should be of the form `format_args = schema:test foo:bar ...`

consumes=FORMAT.string, produces=FORMAT.string

class `stetl.filters.templatingfilter.Jinja2TemplatingFilter` (*configdict, section*)

Bases: `stetl.filters.templatingfilter.TemplatingFilter`

Implements Templating using Jinja2. Jinja2 <http://jinja.pocoo.org>, is a modern and designer-friendly templating language for Python modelled after Django's templates. A 'struct' format as input provides a tree-like structure that could originate from a JSON file or REST service. This input struct provides all the variables to be inserted into the template. The template itself can be configured in this component as a Jinja2 string or -file. An optional 'template_search_paths' provides a list of directories from which templates can be fetched. Default is the current working directory. Via the optional 'globals_path' a JSON structure can be inserted into the Template environment. The variables in this globals structure are typically "boilerplate" constants like: id-prefixes, point of contacts etc.

consumes=FORMAT.struct, produces=FORMAT.string

add_env_filters (*jinja2_env*)

Register additional Filters on the template environment by updating the filters dict: Somehow min and max of list are not present so add them as well.

create_template ()

To be overridden in subclasses.

static `geojson2gml_filter` (*source_crs=4326, target_crs=None, gml_id=None, gml_format='GML2', gml_longsrns='NO'*)

Jinja2 custom Filter: generates any GML geometry from a GeoJSON geometry. By specifying a `target_crs` we can even reproject from the source CRS. The `gml_format=GML2|GML3` determines the general GML form: e.g. `pos/posList` or `coordinates`. `gml_longsrns=YES|NO` determines the `srsName` format like `EPSG:4326` or `urn:ogc:def:crs:EPSG::4326` (long).

template_globals_path ()

CONFIG One or more JSON files or URLs with global variables that can be used anywhere in template. Multiple files will be merged into one globals dictionary

- type: str
- required: False
- default: None

template_search_paths ()

CONFIG List of directories where to search for templates, default is current working directory only.

- type: str
- required: False
- default: None

class `stetl.filters.templatingfilter.StringTemplatingFilter` (*configdict, section*)

Bases: `stetl.filters.templatingfilter.TemplatingFilter`

Implements Templating using Python's internal string.Template. A template string or file should be configured. The input record contains the actual values to be substituted in the template string as a record (key/value pairs). Output is a regular string.

consumes=FORMAT.record or FORMAT.record_array, produces=FORMAT.string

create_template()

To be overridden in subclasses.

safe_substitution()

CONFIG Apply safe substitution? With this method, string.Template.safe_substitute will be invoked, instead of string.Template.substitute. If placeholders are missing from mapping and keywords, instead of raising an exception, the original placeholder will appear in the resulting string intact.

- type: bool
- required: False
- default: False

class stetl.filters.templatingfilter.**TemplatingFilter** (*configdict*, *section*,
consumes='any', *pro-*
duces='string')

Bases: *stetl.filter.Filter*

Abstract base class for specific template-based filters. See <https://wiki.python.org/moin/Templating> Subclasses implement a specific template language like Python string.Template, Mako, Genshi, Jinja2,

consumes=FORMAT.any, produces=FORMAT.string

create_template()

To be overridden in subclasses.

exit()

Allows derived Components to perform a one-time exit/cleanup.

init()

Allows derived Components to perform a one-time init.

invoke(packet)

Components override for Component-specific behaviour, typically read, filter or write actions.

template_file()

CONFIG Path to template file. One of template_file or template_string needs to be configured.

- type: str
- required: False
- default: None

template_string()

CONFIG Template string. One of template_file or template_string needs to be configured.

- type: str
- required: False
- default: None

class `stetl.filters.gmlfeatureextractor.GmlFeatureExtractor` (*configdict*, *section='gml_feature_extractor'*)

Bases: `stetl.filter.Filter`

Extract arrays of GML features etree elements from etree docs.

consumes=FORMAT.etree_doc, produces=FORMAT.etree_feature_array

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.filters.gmlsplitter.GmlSplitter` (*configdict*, *section='gml_splitter'*)

Bases: `stetl.filter.Filter`

Split a stream of text XML lines into documents DEPRECATED: use the more robust XmlElementStreamer-FileInput+XmlAssembler instead!!! TODO phase out

consumes=FORMAT.xml_line_stream, produces=FORMAT.etree_doc

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

class `stetl.filters.formatconverter.FormatConverter` (*configdict*, *section*)

Bases: `stetl.filter.Filter`

Converts (almost) any packet format (if converter available).

consumes=FORMAT.any, produces=FORMAT.any but actual formats are changed at initialization based on the input to output format to be converted via the `input_format` and `output_format` config parameters.

converter_args ()

CONFIG Custom converter-specific arguments.

- type: dict
- required: False
- default: None

static etree_doc2geojson_collection (*converter_args=None*)

Use `converter_args` to determine XML tag names for features and GeoJSON feature id. For example

```
converter_args = { 'root_tag': 'FeatureCollection', 'feature_tag': 'featureMember', 'feature_id_attr': 'fid' }
```

Parameters

- **packet** –
- **converter_args** –

Returns

static etree_doc2struct (*strip_space=True*, *strip_ns=True*, *sub=False*, *attr_prefix=""*, *gml2ogr=True*, *ogr2json=True*)

Parameters

- **packet** –
- **strip_space** –
- **strip_ns** –
- **sub** –

- `attr_prefix` –
- `gml2ogr` –
- `ogr2json` –

Returns

static etree_elem2geojson_feature (*converter_args=None*)

static etree_elem2struct (*strip_space=True, strip_ns=True, sub=False, attr_prefix="", gml2ogr=True, ogr2json=True*)

Parameters

- `packet` –
- `strip_space` –
- `strip_ns` –
- `sub` –
- `attr_prefix` –
- `gml2ogr` –
- `ogr2json` –

Returns

init ()

Allows derived Components to perform a one-time init.

invoke (*packet*)

Components override for Component-specific behaviour, typically read, filter or write actions.

6.5 Components: Outputs

class `stetl.outputs.fileoutput.FileOutput` (*configdict, section*)

Bases: `stetl.output.Output`

Pretty print input to file. Input may be an etree doc or any other stringify-able input.

consumes=FORMAT.any

file_path ()

CONFIG Path to file, for MultiFileOutput can be of the form like: `gmlcities-%03d.gml`

- type: str
- required: True
- default: None

class `stetl.outputs.fileoutput.MultiFileOutput` (*configdict, section*)

Bases: `stetl.outputs.fileoutput.FileOutput`

Print to multiple files from subsequent packets like strings or etree docs, `file_path` must be of a form like: `gmlcities-%03d.gml`.

consumes=FORMAT.any

class `stetl.outputs.standardoutput.StandardOutput` (*configdict, section*)

Bases: `stetl.output.Output`

Print any input to standard output.

consumes=FORMAT.any

class `stetl.outputs.standardoutput.StandardXmlOutput` (*configdict, section*)

Bases: `stetl.output.Output`

Pretty print XML from etree doc to standard output. OBSOLETE, can be done with StandardOutput

consumes=FORMAT.etree_doc

class `stetl.outputs.httpoutput.HttpOutput` (*configdict, section, consumes='any'*)

Bases: `stetl.output.Output`

Output via HTTP protocol, usually via POST.

consumes=FORMAT.any

content_type ()

CONFIG The HTTP ContentType request header for target request.

- type: str
- required: False
- default: text/xml

create_payload (*packet*)

Create a HTTP body payload like for POST of an XML or JSON message. Subclasses like WFS and SOS override. :param packet: :return payload as string:

host ()

CONFIG The hostname/IP addr for target request.

- type: str
- required: True
- default: None

list_fanout ()

CONFIG If we consume a list(), should we create a HTTP req for each member?

- type: bool
- required: False
- default: True

method ()

CONFIG The HTTP method for target request.

- type: str
- required: False
- default: POST

password()

CONFIG The Password for HTTP basic auth for target request.

- type: str
- required: False
- default: None

path()

CONFIG The path number for target request.

- type: str
- required: False
- default: /

port()

CONFIG The port number for target request.

- type: int
- required: False
- default: 80

user()

CONFIG The Username for HTTP basic auth for target request.

- type: str
- required: False
- default: None

class `stetl.outputs.ogroutput.Ogr2OgrOutput` (*configdict, section*)

Bases: `stetl.output.Output`

Output from GML etree doc to any OGR2OGR output using the GDAL/OGR ogr2ogr command

consumes=FORMAT.etree_doc

class `stetl.outputs.ogroutput.OgrOutput` (*configdict, section*)

Bases: `stetl.output.Output`

Direct GDAL OGR output via Python OGR wrapper. Via the Python API <http://gdal.org/python> OGR Features are written.

This output can write almost any geospatial, OGR-defined, dataformat.

consumes=FORMAT.ogr_feature or FORMAT.ogr_feature_array

append()

CONFIG Add to destination destination if it exists (ogr2ogr -append option).

- type: bool
- required: False

- default: False

dest_create_options ()

CONFIG Creation options.

Examples: ..

- type: list
- required: False
- default: []

dest_data_source ()

CONFIG String denoting the OGR data destination. Usually a path to a file like “path/rivers.shp” or connection string to PostgreSQL like “PG: host=localhost dbname='rivers' user='postgres'”.

- type: str
- required: True
- default: None

dest_format ()

CONFIG Instructs GDAL to use driver by that name to open data destination. Not required for many standard formats that are self-describing like ESRI Shapefile.

Examples: ‘PostgreSQL’, ‘GeoJSON’ etc

- type: str
- required: False
- default: None

dest_options ()

CONFIG Custom data destination-specific options. Used in `gdal.SetConfigOption()`.

- type: dict
- required: False
- default: None

init ()

Allows derived Components to perform a one-time init.

layer_create_options ()

CONFIG Options for newly created layer (-lco).

- type: list
- required: False
- default: []

new_layer_name ()

CONFIG Layer name for layer created in the destination source.

- type: str
- required: True
- default: None

overwrite ()

CONFIG Overwrite destination if it exists (ogr2ogr -overwrite option).

- type: bool
- required: False
- default: False

sql ()

CONFIG String with SQL query. Mandatory for PostgreSQL OGR dest.

- type: str
- required: False
- default: None

target_srs ()

CONFIG SRS (projection) for the target.

- type: str
- required: False
- default: None

class `stetl.outputs.execoutput.CommandExecOutput` (*configdict, section*)

Bases: `stetl.outputs.execoutput.ExecOutput`

Executes an arbitrary command.

consumes=FORMAT.string

class `stetl.outputs.execoutput.ExecOutput` (*configdict, section, consumes*)

Bases: `stetl.output.Output`

Executes any command (abstract base class).

class `stetl.outputs.execoutput.Ogr2OgrExecOutput` (*configdict, section*)

Bases: `stetl.outputs.execoutput.ExecOutput`

Executes an Ogr2Ogr command. Input is a file name to be processed. Output by calling Ogr2Ogr command.

consumes=FORMAT.string

cleanup_input ()

CONFIG Flag to indicate whether the input file to ogr2ogr should be cleaned up.

- type: bool
- required: False

- default: False

dest_data_source ()

CONFIG String denoting the OGR data destination. Usually a path to a file like “path/rivers.shp” or connection string to PostgreSQL like “PG: host=localhost dbname='rivers' user='postgres'”.

- type: str
- required: True
- default: None

dest_format ()

CONFIG Instructs GDAL to use driver by that name to open data destination. Not required for many standard formats that are self-describing like ESRI Shapefile.

Examples: ‘PostgreSQL’, ‘GeoJSON’ etc

- type: str
- required: False
- default: None

gfs_template ()

CONFIG Name of GFS template file to use during loading. Passed to ogr2ogr as `-config GML_GFS_TEMPLATE <name>`

- type: str
- required: False
- default: None

lco ()

CONFIG Options for newly created layer (-lco).

- type: str
- required: False
- default: None

options ()

CONFIG Miscellaneous options to pass to ogr2ogr.

- type: str
- required: False
- default: None

spatial_extent ()

CONFIG Spatial extent (-spat), to pass as `xmin ymin xmax ymax`

- type: str

- required: False
- default: None

class `stetl.outputs.dboutput.DbOutput` (*configdict, section, consumes*)
Bases: `stetl.output.Output`

Output to any database (abstract base class).

class `stetl.outputs.dboutput.PostgresDbOutput` (*configdict, section*)
Bases: `stetl.outputs.dboutput.DbOutput`

Output to PostgreSQL database. Input is an SQL string. Output by executing input SQL string.

`consumes=FORMAT.string`

database ()

CONFIG Database name.

- type: str
- required: True
- default: None

host ()

CONFIG Hostname for DB.

- type: str
- required: False
- default: None

password ()

CONFIG DB Password for user.

- type: str
- required: False
- default: None

schema ()

CONFIG Postgres schema name for DB.

- type: str
- required: False
- default: public

user ()

CONFIG DB User name.

- type: str
- required: False

- default: None

class `stetl.outputs.dboutput.PostgresInsertOutput` (*configdict*, *section*, *consumes='record'*)

Bases: `stetl.outputs.dboutput.PostgresDbOutput`

Output by inserting a single record in a Postgres database table. Input is a Stetl record (Python dict structure) or a list of records. Creates an INSERT for Postgres to insert each single record. When the “replace” parameter is True, any existing record keyed by “key” is attempted to be UPDATED first.

NB a constraint is that the first and each subsequent each record needs to contain all values as an INSERT and UPDATE query template is built once for the columns in the first record.

`consumes=[FORMAT.record_array, FORMAT.record]`

exit ()

Allows derived Components to perform a one-time exit/cleanup.

init ()

Allows derived Components to perform a one-time init.

key ()

CONFIG The key column name of the table, required when replacing records.

- type: str
- required: False
- default: None

replace ()

CONFIG Replace record if exists?

- type: bool
- required: False
- default: False

table ()

CONFIG Table for inserts.

- type: str
- required: False
- default: public

class `stetl.outputs.wfsoutput.WFSTOutput` (*configdict*, *section*)

Bases: `stetl.output.Output`

Insert features via WFS-T (WFS Transaction) OGC protocol from an etree doc.

`consumes=FORMAT.etree_doc`

idgen ()

CONFIG Mode that WFS server generates new Id’s for incoming Features.

- type: str

- required: False
- default: GenerateNew

wfs_host ()

CONFIG Hostname-part of URL e.g. geodata.ngr.nl.

- type: str
- required: True
- default: None

wfs_path ()

CONFIG Path-part of URL e.g. '/bag/wfs'.

- type: str
- required: True
- default: None

wfs_port ()

CONFIG Port-part of URL.

- type: str
- required: False
- default: 80

class `stetl.outputs.deegreeoutput.DeegreeBlobstoreOutput` (*configdict, section*)

Bases: `stetl.output.Output`

Insert features into deegree Blobstore from an etree doc.

consumes=FORMAT.etree_doc

init ()

Allows derived Components to perform a one-time init.

class `stetl.outputs.deegreeoutput.DeegreeFSLoaderOutput` (*configdict, section*)

Bases: `stetl.output.Output`

Insert features via deegree using deegree's FSLoader tool from an etree doc.

consumes=FORMAT.etree_doc

CHAPTER 7

Contact

The website stetl.org is the main entry point for all of Stetl.

All development is done via GitHub: see <https://github.com/geopython/stetl>.

Contact the main author Just van den Broecke via email at just@justobjects.nl.

Online chat via Gitter: <https://gitter.im/geopython/stetl>

CHAPTER 8

Links

Below links relevant to Stetl.

CHAPTER 9

Presentations

Below several presentations on Stetl given at various events. The most recent/relevant at the top.

- [GeoPython2016 - Spatial ETL with Stetl](#)
- [5-minute intro Stetl](#)
- [FOSS4G Nottingham 2013](#)
- [Eurogeographics 2013 - INSPIRE Transform with Stetl:](#)
- [Video recording of Eurogeographics 2013 Stetl pres:](#)
- [Several presentations on Stetl on SlideShare \(search for 'Stetl'\)](#)

CHAPTER 10

Stetl Projects/Cases

Known uses of Stetl. More detail in the chapter on *Cases*.

- NLExtract
- SOSpilot
- Smart Emission
- INSPIRE FOSS Project (Archived)

CHAPTER 11

Tools

Tools/components used by/with Stetl.

- GDAL/OGR
- lxml
- deegree WMS/WFS
- PostGIS/PostgreSQL
- Jinja2

CHAPTER 12

Other

- More Geospatial Python projects,
- INSPIRE,

CHAPTER 13

Indices and tables

- `genindex`
- `modindex`
- `search`

S

- stetl.chain, 27
- stetl.component, 26
- stetl.etl, 25
- stetl.factory, 26
- stetl.filter, 28
- stetl.filters.formatconverter, 45
- stetl.filters.gmlfeatureextractor, 44
- stetl.filters.gmlsplitter, 45
- stetl.filters.stringfilter, 42
- stetl.filters.templatingfilter, 43
- stetl.filters.xmlassembler, 42
- stetl.filters.xmlelementreader, 42
- stetl.filters.xmlvalidator, 42
- stetl.filters.xsltfilter, 41
- stetl.input, 28
- stetl.inputs.dbinput, 29
- stetl.inputs.deegreeinput, 40
- stetl.inputs.fileinput, 31
- stetl.inputs.httpinput, 35
- stetl.inputs.ogrinput, 37
- stetl.main, 25
- stetl.merger, 29
- stetl.output, 28
- stetl.outputs.dboutput, 52
- stetl.outputs.deegreeoutput, 54
- stetl.outputs.execoutput, 50
- stetl.outputs.fileoutput, 46
- stetl.outputs.httpoutput, 47
- stetl.outputs.ogroutput, 48
- stetl.outputs.standardoutput, 46
- stetl.outputs.wfsoutput, 53
- stetl.packet, 27
- stetl.splitter, 28

A

add() (stetl.chain.Chain method), 27
 add_authorization() (stetl.inputs.httpinput.HttpInput method), 36
 add_env_filters() (stetl.filters.templatingfilter.Jinja2TemplatingFilter method), 43
 after_chain_invoke() (stetl.component.Component method), 26
 after_chain_invoke() (stetl.splitter.Splitter method), 28
 after_invoke() (stetl.component.Component method), 26
 after_invoke() (stetl.splitter.Splitter method), 28
 ApacheDirInput (class in stetl.inputs.httpinput), 35
 ApacheLogFileInput (class in stetl.inputs.fileinput), 31
 append() (stetl.outputs.ogrouput.OgrOutput method), 48
 assemble() (stetl.chain.Chain method), 27
 auth() (stetl.inputs.httpinput.HttpInput method), 36

B

before_invoke() (stetl.component.Component method), 26
 before_invoke() (stetl.splitter.Splitter method), 28

C

Chain (class in stetl.chain), 27
 class_forname() (stetl.factory.Factory method), 26
 cleanup_input() (stetl.outputs.execoutput.Ogr2OgrExecOutput method), 50
 column_names() (stetl.inputs.dbinput.SqlDbInput method), 30
 CommandExecOutput (class in stetl.outputs.execoutput), 50
 Component (class in stetl.component), 26
 Config (class in stetl.component), 27
 content_type() (stetl.outputs.httpoutput.HttpOutput method), 47
 converter_args() (stetl.filters.formatconverter.FormatConverter method), 45
 create_payload() (stetl.outputs.httpoutput.HttpOutput method), 47

create_template() (stetl.filters.templatingfilter.Jinja2TemplatingFilter method), 43
 create_template() (stetl.filters.templatingfilter.StringTemplatingFilter method), 44
 create_template() (stetl.filters.templatingfilter.TemplatingFilter method), 44
 CsvFileInput (class in stetl.inputs.fileinput), 32

D

data_source() (stetl.inputs.ogrinput.OgrInput method), 37
 database() (stetl.outputs.dboutput.PostgresDbOutput method), 52
 database_name() (stetl.inputs.dbinput.SqlDbInput method), 30
 DbInput (class in stetl.inputs.dbinput), 29
 DbOutput (class in stetl.outputs.dboutput), 52
 DeegreeBlobstoreInput (class in stetl.inputs.deegreeinput), 40
 DeegreeBlobstoreOutput (class in stetl.outputs.deegreeoutput), 54
 DeegreeFSLoaderOutput (class in stetl.outputs.deegreeoutput), 54
 delimiter() (stetl.inputs.fileinput.CsvFileInput method), 32
 depth_search() (stetl.inputs.fileinput.FileInput method), 32
 dest_create_options() (stetl.outputs.ogrouput.OgrOutput method), 49
 dest_data_source() (stetl.outputs.execoutput.Ogr2OgrExecOutput method), 51
 dest_data_source() (stetl.outputs.ogrouput.OgrOutput method), 49
 dest_format() (stetl.outputs.execoutput.Ogr2OgrExecOutput method), 51
 dest_format() (stetl.outputs.ogrouput.OgrOutput method), 49
 dest_options() (stetl.outputs.ogrouput.OgrOutput method), 49
 do_query() (stetl.inputs.dbinput.SqlDbInput method), 30

E

element_tags() (stetl.filters.xmlreader.XmlElementReader static method), 42
 element_tags() (stetl.inputs.fileinput.XmlElementStreamerFileInput method), 34
 end_container() (stetl.inputs.deegreeinput.DeegreeBlobstoreInput method), 40
 end_feature_tag() (stetl.inputs.deegreeinput.DeegreeBlobstoreInput method), 41
 env_expand_args_dict() (stetl.etl.ETL method), 25
 ETL (class in stetl.etl), 25
 etree_doc2geojson_collection() (stetl.filters.formatconverter.FormatConverter static method), 45
 etree_doc2struct() (stetl.filters.formatconverter.FormatConverter static method), 45
 etree_elem2geojson_feature() (stetl.filters.formatconverter.FormatConverter static method), 46
 etree_elem2struct() (stetl.filters.formatconverter.FormatConverter static method), 46
 ExecOutput (class in stetl.outputs.execoutput), 50
 exit() (stetl.component.Component method), 26
 exit() (stetl.filters.templatingfilter.TemplatingFilter method), 44
 exit() (stetl.inputs.dbinput.PostgresDbInput method), 29
 exit() (stetl.outputs.dboutput.PostgresInsertOutput method), 53
 exit() (stetl.splitter.Splitter method), 28

F

Factory (class in stetl.factory), 26
 file_ext() (stetl.inputs.httpinput.ApacheDirInput method), 35
 file_path() (stetl.inputs.fileinput.FileInput method), 33
 file_path() (stetl.outputs.fileoutput.FileOutput method), 46
 FileInput (class in stetl.inputs.fileinput), 32
 filename_pattern() (stetl.inputs.fileinput.FileInput method), 33
 FileOutput (class in stetl.outputs.fileoutput), 46
 Filter (class in stetl.filter), 28
 filter_file() (stetl.inputs.httpinput.ApacheDirInput method), 35
 first() (stetl.merger.Merger method), 29
 FORMAT (class in stetl.packet), 27
 format_args() (stetl.inputs.fileinput.StringFileInput method), 33
 format_data() (stetl.inputs.httpinput.HttpInput method), 36
 FormatConverter (class in stetl.filters.formatconverter), 45

G

Gml2Geojson2gml_filter() (stetl.filters.templatingfilter.Jinja2TemplatingFilter static method), 43
 GmlInput class() (stetl.chain.Chain method), 27
 get_by_id() (stetl.chain.Chain method), 27
 input_index() (stetl.chain.Chain method), 27
 gfs_template() (stetl.outputs.execoutput.Ogr2OgrExecOutput Input method), 51
 GlobFileInput (class in stetl.inputs.fileinput), 33
 GmlFeatureExtractor (class in stetl.filters.gmlfeatureextractor), 44
 GmlSplitter (class in stetl.filters.gmlsplitter), 45

H

host() (stetl.inputs.dbinput.PostgresDbInput method), 29
 host() (stetl.outputs.dboutput.PostgresDbOutput method), 52
 host() (stetl.outputs.httpoutput.HttpOutput method), 47
 HttpInput (class in stetl.inputs.httpinput), 35
 HttpOutput (class in stetl.outputs.httpoutput), 47

I

idgen() (stetl.outputs.wfsoutput.WFSTOutput method), 53
 in_pg_db() (stetl.inputs.ogrinput.OgrPostgisInput method), 38
 in_pg_host() (stetl.inputs.ogrinput.OgrPostgisInput method), 38
 in_pg_password() (stetl.inputs.ogrinput.OgrPostgisInput method), 39
 in_pg_port() (stetl.inputs.ogrinput.OgrPostgisInput method), 39
 in_pg_schema() (stetl.inputs.ogrinput.OgrPostgisInput method), 39
 in_pg_sql() (stetl.inputs.ogrinput.OgrPostgisInput method), 39
 in_pg_user() (stetl.inputs.ogrinput.OgrPostgisInput method), 39
 in_srs() (stetl.inputs.ogrinput.OgrPostgisInput method), 39
 init() (stetl.component.Component method), 26
 init() (stetl.filters.formatconverter.FormatConverter method), 46
 init() (stetl.filters.templatingfilter.TemplatingFilter method), 44
 init() (stetl.inputs.dbinput.PostgresDbInput method), 29
 init() (stetl.inputs.dbinput.SqliteDbInput method), 31
 init() (stetl.inputs.deegreeinput.DeegreeBlobstoreInput method), 41
 init() (stetl.inputs.fileinput.CsvFileInput method), 32
 init() (stetl.inputs.httpinput.ApacheDirInput method), 35
 init() (stetl.inputs.ogrinput.OgrInput method), 38
 init() (stetl.inputs.ogrinput.OgrPostgisInput method), 40

- init() (stetl.outputs.dboutput.PostgresInsertOutput method), 53
 init() (stetl.outputs.deegreeoutput.DeegreeBlobstoreOutput method), 54
 init() (stetl.outputs.ogroutput.OgrOutput method), 49
 init() (stetl.splitter.Splitter method), 29
 Input (class in stetl.input), 28
 input_format() (stetl.component.Component method), 26
 invoke() (stetl.component.Component method), 27
 invoke() (stetl.filter.Filter method), 28
 invoke() (stetl.filters.formatconverter.FormatConverter method), 46
 invoke() (stetl.filters.gmlfeatureextractor.GmlFeatureExtractor method), 45
 invoke() (stetl.filters.gmlsplitter.GmlSplitter method), 45
 invoke() (stetl.filters.stringfilter.StringFilter method), 42
 invoke() (stetl.filters.templatingfilter.TemplatingFilter method), 44
 invoke() (stetl.filters.xmlassembler.XmlAssembler method), 42
 invoke() (stetl.filters.xmlelementreader.XmlElementReader method), 42
 invoke() (stetl.filters.xmlvalidator.XmlSchemaValidator method), 42
 invoke() (stetl.filters.xsltfilter.XsltFilter method), 41
 invoke() (stetl.input.Input method), 28
 invoke() (stetl.output.Output method), 28
 invoke() (stetl.splitter.Splitter method), 29
- ## J
- Jinja2TemplatingFilter (class in stetl.filters.templatingfilter), 43
 JsonFileInput (class in stetl.inputs.fileinput), 33
- ## K
- key() (stetl.outputs.dboutput.PostgresInsertOutput method), 53
 key_map() (stetl.inputs.fileinput.ApacheLogFileInput method), 31
- ## L
- last() (stetl.merger.Merger method), 29
 layer_create_options() (stetl.outputs.ogroutput.OgrOutput method), 49
 lco() (stetl.outputs.execoutput.Ogr2OgrExecOutput method), 51
 LineStreamerFileInput (class in stetl.inputs.fileinput), 33
 list_fanout() (stetl.outputs.httpoutput.HttpOutput method), 47
 log_format() (stetl.inputs.fileinput.ApacheLogFileInput method), 32
- ## M
- main() (in module stetl.main), 25
- max_features_per_doc() (stetl.inputs.deegreeinput.DeegreeBlobstoreInput method), 41
 Merger (class in stetl.merger), 29
 method() (stetl.outputs.httpoutput.HttpOutput method), 47
 MultiFileOutput (class in stetl.outputs.fileoutput), 46
- ## N
- name_filter() (stetl.inputs.fileinput.ZipFileInput method), 35
 new_instance() (stetl.factory.Factory method), 26
 new_layer_name() (stetl.outputs.ogroutput.OgrOutput method), 49
 next_file() (stetl.inputs.httpinput.ApacheDirInput method), 35
 no_more_files() (stetl.inputs.httpinput.ApacheDirInput method), 35
- ## O
- Ogr2OgrExecOutput (class in stetl.outputs.execoutput), 50
 Ogr2OgrOutput (class in stetl.outputs.ogroutput), 48
 OgrInput (class in stetl.inputs.ogrinput), 37
 OgrOutput (class in stetl.outputs.ogroutput), 48
 OgrPostgisInput (class in stetl.inputs.ogrinput), 38
 options() (stetl.outputs.execoutput.Ogr2OgrExecOutput method), 51
 out_dimension() (stetl.inputs.ogrinput.OgrPostgisInput method), 40
 out_geotype() (stetl.inputs.ogrinput.OgrPostgisInput method), 40
 out_gml_format() (stetl.inputs.ogrinput.OgrPostgisInput method), 40
 out_layer_name() (stetl.inputs.ogrinput.OgrPostgisInput method), 40
 out_srs() (stetl.inputs.ogrinput.OgrPostgisInput method), 40
 Output (class in stetl.output), 28
 output_format() (stetl.component.Component method), 27
 overwrite() (stetl.outputs.ogroutput.OgrOutput method), 50
- ## P
- Packet (class in stetl.packet), 28
 parameters() (stetl.inputs.httpinput.HttpInput method), 36
 password() (stetl.inputs.dbinput.PostgresDbInput method), 29
 password() (stetl.outputs.dboutput.PostgresDbOutput method), 52
 password() (stetl.outputs.httpoutput.HttpOutput method), 47
 path() (stetl.outputs.httpoutput.HttpOutput method), 48
 port() (stetl.inputs.dbinput.PostgresDbInput method), 30

- port() (stetl.outputs.httpoutput.HttpOutput method), 48
- PostgresDbInput (class in stetl.inputs.dbinput), 29
- PostgresDbOutput (class in stetl.outputs.dboutput), 52
- PostgresInsertOutput (class in stetl.outputs.dboutput), 53
- print_doc() (in module stetl.main), 25
- process_line() (stetl.inputs.fileinput.LineStreamerFileInput method), 33
- ## Q
- query() (stetl.inputs.dbinput.SqlDbInput method), 31
- quote_char() (stetl.inputs.fileinput.CsvFileInput method), 32
- ## R
- raw_query() (stetl.inputs.dbinput.PostgresDbInput method), 30
- raw_query() (stetl.inputs.dbinput.SqlDbInput method), 31
- raw_query() (stetl.inputs.dbinput.SqliteDbInput method), 31
- read() (stetl.inputs.httpinput.ApacheDirInput method), 35
- read() (stetl.inputs.httpinput.HttpInput method), 37
- read_file() (stetl.inputs.fileinput.FileInput method), 33
- read_file() (stetl.inputs.fileinput.JsonFileInput method), 33
- read_file() (stetl.inputs.fileinput.StringFileInput method), 34
- read_file() (stetl.inputs.fileinput.XmlFileInput method), 34
- read_from_url() (stetl.inputs.httpinput.HttpInput method), 37
- read_once() (stetl.inputs.dbinput.SqlDbInput method), 31
- replace() (stetl.outputs.dboutput.PostgresInsertOutput method), 53
- result_to_output() (stetl.inputs.dbinput.SqlDbInput method), 31
- run() (stetl.chain.Chain method), 27
- ## S
- safe_substitution() (stetl.filters.templatingfilter.StringTemplatingFilter method), 44
- schema() (stetl.inputs.dbinput.PostgresDbInput method), 30
- schema() (stetl.outputs.dboutput.PostgresDbOutput method), 52
- source_format() (stetl.inputs.ogrinput.OgrInput method), 38
- source_options() (stetl.inputs.ogrinput.OgrInput method), 38
- spatial_extent() (stetl.outputs.execoutput.Ogr2OgrExecOutput method), 51
- Splitter (class in stetl.splitter), 28
- sql() (stetl.inputs.ogrinput.OgrInput method), 38
- sql() (stetl.outputs.ogroutput.OgrOutput method), 50
- SqlDbInput (class in stetl.inputs.dbinput), 30
- SqliteDbInput (class in stetl.inputs.dbinput), 31
- StandardOutput (class in stetl.outputs.standardoutput), 46
- StandardXmlOutput (class in stetl.outputs.standardoutput), 47
- start_container() (stetl.inputs.deegreeinput.DeegreeBlobstoreInput method), 41
- start_feature_tag() (stetl.inputs.deegreeinput.DeegreeBlobstoreInput method), 41
- stetl.chain (module), 27
- stetl.component (module), 26
- stetl.etl (module), 25
- stetl.factory (module), 26
- stetl.filter (module), 28
- stetl.filters.formatconverter (module), 45
- stetl.filters.gmlfeatureextractor (module), 44
- stetl.filters.gmlsplitter (module), 45
- stetl.filters.stringfilter (module), 42
- stetl.filters.templatingfilter (module), 43
- stetl.filters.xmlassembler (module), 42
- stetl.filters.xmlelementreader (module), 42
- stetl.filters.xmlvalidator (module), 42
- stetl.filters.xsltfilter (module), 41
- stetl.input (module), 28
- stetl.inputs.dbinput (module), 29
- stetl.inputs.deegreeinput (module), 40
- stetl.inputs.fileinput (module), 31
- stetl.inputs.httpinput (module), 35
- stetl.inputs.ogrinput (module), 37
- stetl.main (module), 25
- stetl.merger (module), 29
- stetl.output (module), 28
- stetl.outputs.dboutput (module), 52
- stetl.outputs.deegreeoutput (module), 54
- stetl.outputs.execoutput (module), 50
- stetl.outputs.fileoutput (module), 46
- stetl.outputs.httpoutput (module), 47
- stetl.outputs.ogroutput (module), 48
- stetl.outputs.standardoutput (module), 46
- stetl.outputs.wfsoutput (module), 53
- stetl.packet (module), 27
- stetl.splitter (module), 28
- StringFileInput (class in stetl.inputs.fileinput), 33
- StringFilter (class in stetl.filters.stringfilter), 42
- StringSubstitutionFilter (class in stetl.filters.stringfilter), 42
- StringTemplatingFilter (class in stetl.filters.templatingfilter), 43
- strip_namespaces() (stetl.filters.xmlelementreader.XmlElementReader method), 42
- strip_namespaces() (stetl.inputs.fileinput.XmlElementStreamerFileInput method), 34

T

- table() (stetl.inputs.dbinput.SqlDbInput method), 31
- table() (stetl.outputs.dboutput.PostgresInsertOutput method), 53
- target_srs() (stetl.outputs.ogroutput.OgrOutput method), 50
- template_file() (stetl.filters.templatingfilter.TemplatingFilter method), 44
- template_globals_path() (stetl.filters.templatingfilter.Jinja2TemplatingFilter method), 43
- template_search_paths() (stetl.filters.templatingfilter.Jinja2TemplatingFilter method), 43
- template_string() (stetl.filters.templatingfilter.TemplatingFilter method), 44
- TemplatingFilter (class in stetl.filters.templatingfilter), 44
- tuples_to_records() (stetl.inputs.dbinput.SqlDbInput method), 31

U

- url() (stetl.inputs.httpinput.HttpInput method), 37
- user() (stetl.inputs.dbinput.PostgresDbInput method), 30
- user() (stetl.outputs.dboutput.PostgresDbOutput method), 52
- user() (stetl.outputs.httpoutput.HttpOutput method), 48

W

- wfs_host() (stetl.outputs.wfsoutput.WFSTOutput method), 54
- wfs_path() (stetl.outputs.wfsoutput.WFSTOutput method), 54
- wfs_port() (stetl.outputs.wfsoutput.WFSTOutput method), 54
- WFSTOutput (class in stetl.outputs.wfsoutput), 53

X

- XmlAssembler (class in stetl.filters.xmlassembler), 42
- XmlElementReader (class in stetl.filters.xmlelementreader), 42
- XmlElementStreamerFileInput (class in stetl.inputs.fileinput), 34
- XmlFileInput (class in stetl.inputs.fileinput), 34
- XmlLineStreamerFileInput (class in stetl.inputs.fileinput), 34
- XmlSchemaValidator (class in stetl.filters.xmlvalidator), 42
- XsltFilter (class in stetl.filters.xsltfilter), 41

Z

- ZipFileInput (class in stetl.inputs.fileinput), 34