<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Installation</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Example Usage</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Sphinx AutoAPI Index</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>birdy</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>birdy.esgsearch</td>
<td>8</td>
</tr>
<tr>
<td>3.3</td>
<td>birdy.utils</td>
<td>8</td>
</tr>
<tr>
<td>3.4</td>
<td>birdy.wpsparser</td>
<td>9</td>
</tr>
</tbody>
</table>

Python Module Index  11
**Birdy (the bird)** *Birdy is not a bird but likes to play with them.*

Birdy is a Python package to provide a command-line tool to work with Web Processing Services (WPS). It is using OWSLib from the GeoPython project.

Birdy is part of the Birdhouse project.

Contents:
Installation

Check out code from the birdy github repo and start the installation:

```sh
$ git clone https://github.com/bird-house/birdy.git
$ cd birdy
$ make install
```

Or install the anaconda package:

```sh
$ conda install -c birdhouse birdhouse-birdy
```
Example Usage

Show the processes of a Web Processing Service:

```bash
$ export WPS_SERVICE=http://localhost:8094/wps
$ birdy -h
usage: birdy [<options>] <command> [<args>]

Emu: WPS processes for testing and demos.

optional arguments:
  -h, --help              show this help message and exit
  --debug                enable debug mode
  --token TOKEN, -t TOKEN
                        Token to access the WPS service.

command:
List of available commands (wps processes)

{helloworld, ultimatequestionprocess, wordcount, inout, multiplesources, chomsky, zonal_mean}

Run "birdy <command> -h" to get additional help.

helloworld    Hello World: Welcome user and say hello ...
ultimatequestionprocess
Answer to Life, the Universe and Everything: Numerical solution that is the answer to Life, Universe and Everything. The process is an improvement to Deep Tought computer (therefore version 2.0) since it no longer takes 7.5 million years, but only a few seconds to give a response, with an update of status every 10 seconds.
wordcount     Word Counter: Counts words in a given text ...
inout         Testing all Data Types: Just testing data types like date, datetime etc ...
multiplesources Multiple Sources: Process with multiple different sources ...
chomsky       Chomsky text generator: Generates a random chomsky text ...
zonal_mean    Zonal Mean: zonal mean in NetCDF File.
```

Show help for wordcount:

```bash
$ birdy wordcount -h
usage: birdy wordcount [-h] --text [TEXT] [--output [{output} [{output} ...]]]
```
optional arguments:
-h, --help            show this help message and exit
--text [TEXT]        Text document: URL of text document, mime
                      types=text/plain
--output {...output}  Output: output=Word count result, mime
                      types=text/plain (default: all outputs)

Execute wordcount with a remote text document:

INFO:Execution status: ProcessAccepted
INFO:Execution status: ProcessSucceeded
INFO:Output:
INFO:output=http://localhost:8090/wpsoutputs/emu/output-7becb14c-41c6-11e5-ae23-68f72837e1b4.txt (text/plain)

The result output is given as a reference document.

You can also use a local file as input document:

INFO:Execution status: ProcessAccepted
INFO:Execution status: ProcessSucceeded
INFO:Output:
INFO:output=http://localhost:8090/wpsoutputs/emu/output-f65f5358-41c6-11e5-ae23-68f72837e1b4.txt (text/plain)

If you run this process on a remote WPS service then local files will be send inline and base64 encoded with the WPS execute request. Please use in this case small files only (a few megabytes)!

If the WPS service is secured by a Twitcher security proxy service then you can provide an access token with the --token option:

3.1 birdy

birdy._wps(url)
birdy.main()

3.1.1 Birdy

class birdy.Birdy

Imports

• object

Summary

Birdy is a command line client for Web Processing Services.
Documentation is available on readthedocs: http://birdy.readthedocs.org/en/latest/
see help: $ birdy -h

__complex_value__(self, key, value)
__input_value__(self, key, value)
__literal_value__(self, key, value)
build_command__(self, subparser, identifier)
create_parser__(self)

Generates parser to execute WPS processes on the command line.

lazy parsing subparser: http://stackoverflow.com/questions/22742450/argparse-on-demand-imports-for-types-choices-etc

execute(self, args)

monitor(self, execution)

Convenience method to monitor the status of a WPS execution till it completes (successfull or not), and write the output to file after a succesfull job completion.

execution: WPSExecution instance
sleepSecs: number of seconds to sleep in between check status invocations
download: True to download the output when the process terminates, False otherwise
filepath: optional path to output file (if downloaded=True), otherwise filepath will be inferred from response document

3.2 birdy.esgsearch

birdy.esgsearch.esgf_search_experiments(prefix, parsed_args)
birdy.esgsearch.esgf_search_projects(prefix, parsed_args)
birdy.esgsearch.esgf_search_variables(prefix, parsed_args)
birdy.esgsearch.main()

3.2.1 ESGSearch

class birdy.esgsearch.ESGSearch

Imports

• object

Summary

Command line client for esgf search.

$ esgsearch -h

create_parser(self)
Generates parser to query esgf search on the command line.

3.3 birdy.utils

birdy.utils.encode(url, mimetypes)
Read file with given url and return content. If mimetype of file is binary then encode content with base64.

If url is not a file:// url return url itself.

Returns encoded content string or URL or None

birdy.utils.fix_local_url(url)
If url is just a local path name then create a file:// URL. Otherwise return url just as it is.

birdy.utils.is_file_url(url)
3.4 birdy.wpsparser

birdy.wpsparser.\texttt{is\_bbox\_data}(\textit{inoutput})
birdy.wpsparser.\texttt{is\_complex\_data}(\textit{inoutput})
birdy.wpsparser.\texttt{parse\_choices}(\textit{input})
birdy.wpsparser.\texttt{parse\_default}(\textit{input})
birdy.wpsparser.\texttt{parse\_description}(\textit{input})
birdy.wpsparser.\texttt{parse\_nargs}(\textit{input})
birdy.wpsparser.\texttt{parse\_process\_help}(\textit{process})
birdy.wpsparser.\texttt{parse\_required}(\textit{input})
birdy.wpsparser.\texttt{parse\_type}(\textit{input})
birdy.wpsparser.\texttt{parse\_wps\_description}(\textit{wps})
b
birdy, 7
birdy.esgsearch, 8
birdy.utils, 8
birdy.wpsparser, 9
Symbols

_wps() (in module birdy), 7

B
Birdy (class in birdy), 7
birdy (module), 7
Birdy._complex_value() (in module birdy), 7
Birdy._input_value() (in module birdy), 7
Birdy._literal_value() (in module birdy), 7
Birdy.build_command() (in module birdy), 7
Birdy.create_parser() (in module birdy), 7
birdy.esgsearch (module), 8
Birdy.execute() (in module birdy), 8
Birdy.monitor() (in module birdy), 8
birdy.utils (module), 8
birdy.wpsparser (module), 9

E
encode() (in module birdy.utils), 8
esgf_search_experiments() (in module birdy.esgsearch), 8
esgf_search_projects() (in module birdy.esgsearch), 8
esgf_search_variables() (in module birdy.esgsearch), 8
ESGSearch (class in birdy.esgsearch), 8
ESGSearch.create_parser() (in module birdy.esgsearch), 8

F
fix_local_url() (in module birdy.utils), 8

I
is_bbox_data() (in module birdy.wpsparser), 9
is_complex_data() (in module birdy.wpsparser), 9
is_file_url() (in module birdy.utils), 8

M
main() (in module birdy), 7
main() (in module birdy.esgsearch), 8

P
parse_choices() (in module birdy.wpsparser), 9
parse_default() (in module birdy.wpsparser), 9
parse_description() (in module birdy.wpsparser), 9
parse_nargs() (in module birdy.wpsparser), 9
parse_process_help() (in module birdy.wpsparser), 9
parse_required() (in module birdy.wpsparser), 9
parse_type() (in module birdy.wpsparser), 9
parse_wps_description() (in module birdy.wpsparser), 9