ARCCSSive Documentation

Release 0.3.3.dev6+g07cd22

ARCCSS CMS

Mar 07, 2018

Contents

ARCCSSive is a Python library developed by the CMS team at the ARC Centre of Excellence for Climate Systems Science for working with data at NCI. Contents:

CHAPTER 1

Installing

1.1 Raijin

The stable version of ARCCSSive is available as a module on NCI's Raijin supercomputer:

```
raijin $ module use ~access/modules
raijin $ module load pythonlib/ARCCSSive
```

1.2 NCI Virtual Desktops

NCI's virtual desktops allow you to use ARCCSSive from a Jupyter notebook. For details on how to use virtual desktops see http://vdi.nci.org.au/help

To install the stable version of ARCCSSive:

```
vdi $ pip install --user ARCCSSive
vdi $ export CMIP5_DB=sqlite:////g/data1/ua6/unofficial-ESG-replica/tmp/tree/cmip5_
↪raijin_latest.db
```

or to install the current development version (note this uses a different database):

```
vdi $ pip install --user git+https://github.com/coecms/ARCCSSive.git
vdi $ export CMIP5_DB=sqlite:////g/data1/ua6/unofficial-ESG-replica/tmp/tree/cmip5_
↪raijin_latest.db
```

Once the library is installed run `ipython notebook` to start a new notebook

CHAPTER 2

CMIP5

The CMIP5 module provides tools for searching through the CMIP5 data stored on NCI's */g/data* filesystem

2.1 Getting Started:

The ARCCSSive library is available as a module on Raijin. Load it using:

```
module use ~access/modules
module load pythonlib/ARCCSSive
```

To use the CMIP5 catalog you first need to connect to it:

```
>>> from ARCCSSive import CMIP5
>>> cmip5 = CMIP5.connect()
```

The session object allows you to run queries on the catalog. There are a number of helper functions for common operations, for instance searching through the model outputs:

```
>>> outputs = cmip5.outputs(
...     experiment = 'rcp45',
...     variable   = 'tas',
...     mip        = 'Amon')
```

You can then loop over the search results in normal Python fashion:

```
>>> for o in outputs:
...     six.print_(o.model, *o.filenames())
ACCESS1-3 example.nc
```

2.2 Examples

2.2.1 Get files from a single model variable

```
>>> outputs = cmip5.outputs(
...     experiment = 'rcp45',
...     variable   = 'tas',
...     mip        = 'Amon',
...     model      = 'ACCESS1-3',
...     ensemble   = 'r1i1p1')

>>> for f in outputs.first().filenames():
...     six.print_(f)
example.nc
```

2.2.2 Get files from all models for a specific variable

```
>>> outputs = cmip5.outputs(
...     experiment = 'rcp45',
...     variable   = 'tas',
...     mip        = 'Amon',
...     ensemble   = 'r1i1p1')

>>> for m in outputs:
...     model = m.model
...     files = m.filenames()
```

2.2.3 Choose more than one variable at a time

More complex queries on the *Session.outputs()* results can be performed using SQLalchemy's filter():

```
>>> from ARCCSSive.CMIP5.Model import *
>>> from sqlalchemy import *

>>> outputs = cmip5.outputs(
...     experiment = 'rcp45',
...     model      = 'ACCESS1-3',
...     mip        = 'Amon',) \
...     .filter(Instance.variable.in_(['tas','pr']))
```

2.2.4 Get results from a specific output version

Querying specific versions currently needs to go through the *Session.query()* function, this will be simplified in a future version of ARCCSSive:

```
>>> from ARCCSSive.CMIP5.Model import *

>>> res = cmip5.query(Version) \
...         .join(Instance) \
...         .filter(
...     Version.version     == 'v20120413',
```

```
...     Instance.model      == 'ACCESS1-3',
...     Instance.experiment == 'rcp45',
...     Instance.mip        == 'Amon',
...     Instance.ensemble   == 'r1i1p1')

>>> # This returns a sequence of Version, get the variable information from
>>> # the .variable property
>>> for o in res:
...     six.print_(o.variable.model, o.variable.variable, o.filenames())
```

2.2.5 Compare model results between two experiments

Link two sets of outputs together using joins:

```
>>> from ARCCSSive.CMIP5.Model import *
>>> from sqlalchemy.orm import aliased
>>> from sqlalchemy import *

>>> # Create aliases for the historical and rcp variables, so we can
>>> # distinguish them in the query
>>> histInstance = aliased(Instance)
>>> rcpInstance  = aliased(Instance)
>>> rcp_hist  = cmip5.query(rcpInstance, histInstance).join(
...         histInstance, and_(
...             histInstance.variable == rcpInstance.variable,
...             histInstance.model    == rcpInstance.model,
...             histInstance.mip      == rcpInstance.mip,
...             histInstance.ensemble == rcpInstance.ensemble,
...         )).filter(
...             rcpInstance.experiment  == 'rcp45',
...             histInstance.experiment == 'historicalNat',
...         )

>>> for r, h in rcp_hist:
...     six.print_(r.versions[-1].path, h.versions[-1].path)
```

2.3 API

2.3.1 connect()

ARCCSSive.CMIP5.**connect**()

Connect to the CMIP5 catalog

Returns A new *Session*

Example:

```
>>> from ARCCSSive import CMIP5
>>> cmip5   = CMIP5.DB.connect()
>>> outputs = cmip5.query()
```

2.3.2 Session

The session object has a number of helper functions for getting information out of the catalog, e.g. *Session.models()* gets a list of all available models.

class ARCCSSive.CMIP5.**Session**
Holds a connection to the catalog

Create using *ARCCSSive.CMIP5.connect()*

experiments()
Get the list of all experiments in the dataset

Returns A list of strings

files(***kwargs*)
Query the list of files

Returns a list of files that match the arguments

Parameters ****kwargs** – Match any attribute in *Model.Instance*, e.g. *model = 'ACCESS1-3'*

Returns An iterable returning `Model.File` matching the search query

mips()
Get the list of all MIP tables in the dataset

Returns A list of strings

models()
Get the list of all models in the dataset

Returns A list of strings

outputs(***kwargs*)
Get the most recent instances matching a query

Arguments are optional, using them will select only matching outputs

Parameters

- **variable** – CMIP variable name
- **experiment** – CMIP experiment
- **mip** – MIP table
- **model** – Model used to generate the dataset
- **ensemble** – Ensemble member

Returns An iterable sequence of *ARCCSSive.CMIP5.Model.Instance*

query(**args*, ***kwargs*)
Query the CMIP5 catalog

Allows you to filter the full list of CMIP5 outputs using SQLAlchemy commands

Returns A SQLalchemy query object

variables()
Get the list of all variables in the dataset

Returns A list of strings

2.3.3 Model

The model classes hold catalog information for a single entry. Each model run variable can have a number of different data versions, as errors get corrected by the publisher, and each version can consist of a number of files split into a time sequence.

class ARCCSSive.CMIP5.Model.**Instance**(***kwargs*)

A model variable from a specific run

Search through these using *ARCCSSive.CMIP5.Session.outputs()*

variable
Variable name

experiment
CMIP experiment

mip
MIP table specifying output frequency and realm

model
Model that generated the dataset

ensemble
Ensemble member

realm
Realm: ie atmos, ocean

versions
List of *Version* available for this output

latest()
Returns latest version/s available on raijin, first check in any version is_latest, then checks date stamp

filenames()
Returns the file names from the latest version of this variable

Returns List of file names

drstree_path()
Returns the drstree path for this instance latest version

class ARCCSSive.CMIP5.Model.**Version**(***kwargs*)

A version of a model run's variable

version
Version identifier

path
Path to the output directory

variable
`Variable` associated with this version

warnings
List of `VersionWarning` available for this output

files
List of `VersionFile` available for this output

```
>>> version = cmip5.query(Version).first()
```

glob()
Get the glob string matching the CMIP5 filename

```
>>> six.print_(version.glob())
a_6hrLev_c_d_e*.nc
```

build_filepaths()
Returns the list of files matching this version

Returns List of file names

```
>>> version.build_filepaths()
[]
```

filenames()
Returns the list of filenames for this version

Returns List of file names

```
>>> version.filenames()
[]
```

tracking_ids()
Returns the list of tracking_ids for files in this version

Returns List of tracking_ids

```
>>> version.tracking_ids()
[]
```

drstree_path()
Returns the drstree path for this particular version

CHAPTER 3

Administration

— Making a new release —

Use the Github interface to create a new relase with the version number, e.g. '1.2.3'. This should use semantic versioning, if it's a minor change increase the third number, if it introduces new features increase the second number and if it will break existing scripts using the library increase the first number.

After doing this the following will happen:

- Travis-ci will upload the package to PyPI
- CircleCI will upload the package to Anaconda
- The conda update cron job at NCI will pick up the new version overnight

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

W