drslib Documentation

Release 0.3.1

Stephen Pascoe

October 14, 2014

Contents

Drslib is a Python library for processing data intended for publishing into the Earth System Grid Federation (ESGF). It was first designed for the 5th Climate Model Intercomparison Project (CMIP5) Data Reference Syntax (DRS). It has subsequently been extended to support the SPECS and CORDEX projects and provides an internal API for future extension. Drslib API-level code for working with DRS components, algorithms for decuding DRS components from incomplete information and a command-line tool for manipulating data files into the recommended DRS directory structure.It has been developed by the Centre for Environmental Data Archival as part of the ESG Federation.

CHAPTER 1

User Guide

1.1 Introduction

1.1.1 Use Cases

The design of the library has been driven by the following requirements:

1. Deduce the DRS directory structure a DRS-compliant filename.
2. Validate filenames and directory paths against the DRS.
3. Convert the CMIP3 directory structure into a DRS-compliant form.
4. Manage multiple versions of DRS publication-level datasets on the filesystem.
5. Detect the CMIP5 product DRS component during data publication.

1.1.2 Installation

drslib is written in Python. It requires one of the installation tools setuptools or pip. Other requirements are downloaded automatically during installation. Download and install with:

```
# Using pip
$ pip install drslib

# OR using setuptools
$ easy_install drslib
```

If you are upgrading drslib you will need to add the "-U" option to either pip or easy_install.

> **Note:** To install onto an ESG datanode you can use use the `easy_install` in `/usr/local/uv-cdat/bin`. This will install `drslib` into `/usr/local/uv-cdat/bin/python` and install `drs_tool` as `/usr/local/uv-cdat/bin/drs_tool`.

Installing for development

The source of drslib is available on github from the ESGF github repo.

Once you have the code you must activate the distribution in *develop* mode. To do this execute the following using a python interpreter with setuptools installed:

```
$ python setup.py develop
```

1.1.3 Configuration

drslib will use CMIP5 MIP tables to deduce realm and other metadata. MIP tables are not required for other projects as selected with the -s scheme switch. You can point to your MIP table location using the environment variable MIP_TABLE_PATH. All tables should be named CMIP5_*.

Alternatively you can use metaconfig to configure the location of your MIP tables. Create the following in your $HOME/.metaconfig.conf file:

```
[metaconfig]
configs = drslib

[drslib:tables]
path = /path/to/mip/tables
model_table = /path/to/model_table
```

The option model_table is optional and if specified should point to the "CMIP5 Modelling Group" spreadsheet in CSV format. This table is used to map model names to institute names. The drslib package is distributed with a recent version of this file.

Default DRS attributes used by the drs_tool command can be set in the drs section:

```
[drslib:drs]
root = /path/to/drs/activity
institute = MOHC
model = HadCM3
```

It is usually convenient to set at least root and activity in the configuration file.

Overriding DRS vocabularies

Drslib ships with reasonable defaults for the DRS vocabularies, however sometimes you will want to override or extend the defaults supplied. At this time drslib supports extending the vocabularies of institutes, models and experiments.

To define all CMIP5 experiments including individual decadal experiments you can define the drslib:vocabularies section as follows:

```
[drslib:vocabularies]
experiments =
  1pctto2x 2xco2 pdcntrl sresa1b 1pctto4x amip picntrl sresa2 20c3m commit
  slabcntl sresb1
  decadal1960 decadal1961 decadal1962 decadal1963 decadal1964 decadal1965
  decadal1966 decadal1967 decadal1968 decadal1969 decadal1970 decadal1971
  decadal1972 decadal1973 decadal1974 decadal1975 decadal1976 decadal1977
  decadal1978 decadal1979 decadal1980 decadal1981 decadal1982 decadal1983
  decadal1984 decadal1985 decadal1986 decadal1987 decadal1988 decadal1989
  decadal1990 decadal1991 decadal1992 decadal1993 decadal1994 decadal1995
  decadal1996 decadal1997 decadal1998 decadal1999 decadal2000 decadal2001
  decadal2002 decadal2003 decadal2004 decadal2005 decadal2006 decadal2007
  decadal2008 decadal2009
```

Institutes and models can be defined with the institutes option. This is interpreted as a newline separated list of lines, each line being the institute name a colon then space separated list of models.

```
institutes =
  NOAA-GFDL:GFDL-ESM2G
  MOHC:HadGEM2-ES HadCM3 HadGEM2-CC
```

1.1.4 Logging

`drslib` uses Python's standard logging infrastructure to give details of it's operation. Messages are sent to loggers under the `drslib` logger. You can configure logging via metaconfig by pointing to a separate logging configuration file:

```
[metaconfig]
configs = drslib
logging = /path/to/logging.conf
```

The format of `logging.conf` should conform to the Python logging **'configuration file format'__**. An example logging configuration is given below which will log product detection decisions separately from general drslib warnings:

```
#
# Basic logging configuration for drs_tool
#
# This configuration prints product detection decisions to STDERR and logs
# warnings to ./drs_tool.log
#

[loggers]
keys=root,drslib,p_cmip5

[handlers]
keys=drslib_h,p_cmip5_h

[formatters]
keys=f1,f2

#-------------------------------------------------------------------------
# Loggers

# No catch-all logging
[logger_root]
handlers=
level=NOTSET

[logger_drslib]
qualname=drslib
handlers=drslib_h

[logger_p_cmip5]
qualname=drslib.p_cmip5
handlers=p_cmip5_h
propagate=0

#-------------------------------------------------------------------------
# Handlers & Formatters

[handler_drslib_h]
class=FileHandler
args=('./drs_tool.log', )
```

```
formatter=f1
level=INFO

[handler_p_cmip5_h]
class=StreamHandler
args=(sys.stderr, )
formatter=f2
level=INFO

[formatter_f1]
format=%(asctime)s [%(levelname)s] %(name)s: %(message)s
datefmt=

[formatter_f2]
format=[%(levelname)s] %(name)s: %(message)s
```

1.1.5 Testing

drslib ships with a test suite compliant with nose. The suite can be run in various ways. The test suite uses the extension NoseXUnit to produce XML reports of the test results. NoseXUnit will be automatically installed if you run the tests via `setup.py`:

```
$ python setup.py test
```

Or if the depencencies are satisfied you can run all tests with:

```
$ nosetests
```

1.1.6 Reporting Bugs

Please report bugs to the github repo.

1.2 drslib API HowTo

1.2.1 Parsing DRS filenames

The drslib.cmip5.make_translator function takes a path or URL prefix as it's only argument

```
>>> from drslib import cmip5
>>> cmip5_trans = cmip5.make_translator('http://example.com/cmip5')
>>> cmip5_trans.prefix
'http://example.com/cmip5'
```

The DRS class is used to store DRS components. For instance to extract DRS components from a filename do:

```
>>> drs = cmip5_trans.filename_to_drs('tas_Amon_HadCM3_historicalNat_r1_185001-200512.nc')
>>> drs.variable
'tas'
>>> drs.table
'Amon'
>>> drs.model
'HadCM3'
```

```
>>> drs.subset
((1850, 1, None, None, None, None), (2005, 12, None, None, None, None), None)
```

Some DRS components can be deduced from the filename by consulting the CMIP5 MIP tables or using internally configured tables. These are automatically set during translation:

```
>>> drs.institute
'MOHC'
>>> drs.realm
'atmos'
>>> drs.frequency
'mon'
```

Some DRS attributes cannot be deduced from the filename. These must be set in the drs object before you can generate a full DRS path. Use the DRS.is_complete() method to test if the DRS object is completely specified:

```
>>> drs.is_complete()
False
>>> drs.version = 3
>>> drs.product = 'output'
>>> drs.is_complete()
True
```

Once complete you can generate the full DRS path or URL:

```
>>> cmip5_trans.drs_to_filepath(drs)
'http://example.com/cmip5/output/MOHC/HadCM3/historicalNat/mon/atmos/Amon/r1/v3/tas/tas_Amon_HadCM3_h
```

1.2.2 Generating DRS paths and filenames

The DRS class can be used to constructed from scratch like:

```
>>> from drslib.drs import CmipDRS
>>> drs2 = CmipDRS(activity='cmip5', product='requested', model='HadCM3',
...            experiment='historicalNat', variable='tas',
...            ensemble=(1, None, None), version=4,
...            table='3hr', frequency='3hr')

>>> cmip5_trans.drs_to_path(drs2)
'http://example.com/cmip5/requested/MOHC/HadCM3/historicalNat/3hr/atmos/3hr/r1/v4/tas'
```

Extra information is needed to create the filename

```
>>> cmip5_trans.drs_to_file(drs2)
'tas_3hr_HadCM3_historicalNat_r1.nc'
>>> drs2.subset = ((2010, None, None, None, None, None), (2011, None, None, None, None, None), None)
>>> cmip5_trans.drs_to_file(drs2)
'tas_3hr_HadCM3_historicalNat_r1_2010-2011.nc'
```

1.3 `drs_tool` command-line interface

The `drs_tool` command can be used to invoke functions from the `drslib.drs_tree` API. The tool is designed to help ESGF datanode managers to

1. Prepare incoming data for publication, placing files in the DRS directory structure.

2. Manage multiple versions of publication-level datasets to minimise disk usage.
3. Deduce the CMIP5 product into which data should be published according to the rules definied in the CMIP5 experiment definition.

1.3.1 Concepts

DRS root The root of the published DRS directory tree. This is synonimous with the directory representing the DRS activity component.

Incoming directory The directory scanned for data files to be added to the DRS directory tree. This directory is scanned recursively to find NetCDF files in the DRS filename encoding. It defaults to `<drs-root>/output` for compatibility with the output structure of CMOR2. However, the directory names under the incoming directory are not taken into account when deducing the DRS components of files.

DRS pattern Each invocation of `drs_tool` is given a set of DRS component values that define the portion of the DRS space on which it acts.

1.3.2 Usage

Usage: drs_tool [command] [options] [drs-pattern]

command:

list	list publication-level datasets
todo	show file operations pending for the next version
upgrade	make changes to the selected datasets to upgrade to the next version
mapfile	make a mapfile of the selected dataset
history	list all versions of the selected dataset
init	initialise CMIP5 product detection data

drs-pattern: A dataset identifier in '.'-separated notation. Use the '%' to indicate unknown DRS components, e.g. `cmip5.%.MPI-M` will match any product.

Options:

-h, --help show this help message and exit

-R ROOT, --root=ROOT Root directory of the DRS tree

-I INCOMING, --incoming=INCOMING Incoming directory for DRS files. Defaults to <root>/output

-a ACTIVITY, --activity=ACTIVITY Set DRS attribute activity for dataset discovery

-p PRODUCT, --product=PRODUCT Set DRS attribute product for dataset discovery

-i INSTITUTE, --institute=INSTITUTE Set DRS attribute institute for dataset discovery

-m MODEL, --model=MODEL Set DRS attribute model for dataset discovery

-e EXPERIMENT, --experiment=EXPERIMENT Set DRS attribute experiment for dataset discovery

-f FREQUENCY, --frequency=FREQUENCY Set DRS attribute frequency for dataset discovery

-r REALM, --realm=REALM Set DRS attribute realm for dataset discovery

-v VERSION, --version=VERSION Force version upgrades to this version

-P FILE, --profile=FILE Profile the script exectuion into FILE

--detect-product Automatically detect the DRS product of incoming data

-j JSON_DRS, --json-drs=JSON_DRS Use the JSON output from the *ceda-cc* quality control tool to define the incoming set of files and their associated DRS terms.

1.3.3 An Example

This example walks through using `drs_tool` to prepare datasets for publication. It uses some dummy data files based on test runs of the Met Office Hadley Centre HadGEM2-ES model. You can repeat the example from within the drslib source distribution directory.

First generate a dummy set of incoming data files. In this example we assume all incoming files are in a single flat directory, however `drs_tool` will recursively travers the incoming directory to find all DRS-compliant NetCDF files.

```
$ mkdir mohc_eg
$ python test/gen_drs.py test/mohc_delivery.ls mohc_eg/incoming
$ ls mohc_eg/incoming/ | head
baresoilFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_201512-204011.nc
baresoilFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_204012-206511.nc
baresoilFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_206512-209011.nc
baresoilFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_209012-209911.nc
c3PftFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_201512-204011.nc
c3PftFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_204012-206511.nc
c3PftFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_206512-209011.nc
c3PftFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_209012-209911.nc
c4PftFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_201512-204011.nc
c4PftFrac_Lmon_HadGEM2-ES_rcp45_r1i1p1_204012-206511.nc
$ ls mohc_eg/incoming/ | wc -l
494
```

We now have about 500 dummy NetCDF files in `mohc_eg/incoming`. You can ask `drs_tool` to list which publication-level datasets these files would be put in using the `drs_tool list` subcommand. For this to work `drs_tool` requires 2 DRS components not decidable from the filenames: activity and product [1]. `drs_tool list` will list all publication-level datasets with the criteria given, including those that would be created by processing the incoming directory.

```
$ drs_tool list -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1
================================================================================
DRS Tree at mohc_eg/
--------------------------------------------------------------------------------
cmip5.output1.MOHC.HadGEM2-ES.rcp45.3hr.atmos.3hr.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.3hr.land.3hr.r1i1p1                 *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrLev.r1i1p1             *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrPlev.r1i1p1            *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.atmos.day.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.land.day.r1i1p1                 *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.aerosol.aero.r1i1p1             *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.atmos.Amon.r1i1p1               *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.land.Lmon.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.landIce.LImon.r1i1p1            *
================================================================================
```

The asterisk against each dataset_id indicates there are files in the incoming directory to add to the dataset. In this case all datasets are empty.

[1] later versions of drslib will be able to decide the product component from other components and by inspecting the NetCDF.

We can restrict `drs_tool list` output by using a dataset_id wildcard. For instance to select only datasets in the `atmos` realm:

```
$ drs_tool list -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1.%.%.%.%.atmos
================================================================================
DRS Tree at mohc_eg/
--------------------------------------------------------------------------------
cmip5.output1.MOHC.HadGEM2-ES.rcp45.3hr.atmos.3hr.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrLev.r1i1p1             *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrPlev.r1i1p1            *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.atmos.day.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.atmos.Amon.r1i1p1               *
================================================================================
```

The same effect can be achieved with individual component options:

```
$ drs_tool list -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1 --realm=atmos
================================================================================
DRS Tree at mohc_eg/
--------------------------------------------------------------------------------
cmip5.output1.MOHC.HadGEM2-ES.rcp45.3hr.atmos.3hr.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrLev.r1i1p1             *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrPlev.r1i1p1            *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.atmos.day.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.atmos.Amon.r1i1p1               *
================================================================================
```

Now we will focus on a single dataset in the `aerosol` realm and show how to move files into the DRS directory structure ready for publication. We can check what filesystem commands will be done using the `drs_tool todo` subcommand.

```
$ drs_tool list -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1 --realm=aerosol
================================================================================
DRS Tree at mohc_eg/
--------------------------------------------------------------------------------
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.aerosol.aero.r1i1p1              *
================================================================================
$ drs_tool todo -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1 --realm=aerosol | head
================================================================================
DRS Tree at mohc_eg/
--------------------------------------------------------------------------------
Publisher Tree cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.aerosol.aero.r1i1p1 todo for version 20100927
--------------------------------------------------------------------------------
mv mohc_eg/incoming/emidust_aero_HadGEM2-ES_rcp45_r1i1p1_206512-209011.nc /home/spascoe/git/esgf-drsl
ln -s /home/spascoe/git/esgf-drslib/mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/fil
mv mohc_eg/incoming/reffclwtop_aero_HadGEM2-ES_rcp45_r1i1p1_204012-206511.nc /home/spascoe/git/esgf-d
ln -s /home/spascoe/git/esgf-drslib/mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/fil
mv mohc_eg/incoming/dryso2_aero_HadGEM2-ES_rcp45_r1i1p1_204012-206511.nc /home/spascoe/git/esgf-drsli
```

You can see here that drslib will move files into datestamped directories under `<dataset-dir>/files` then symbolically link them into the DRS directory structure. To do the actual moving use `drs_tool upgrade`. Then use `drs_tool list` to view the result.

```
$ drs_tool upgrade -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1 --realm=aerosol
================================================================================
DRS Tree at mohc_eg/
--------------------------------------------------------------------------------
Upgrading cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.aerosol.aero.r1i1p1 to version 20100927 ... done
================================================================================
$ drs_tool list -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1
```

```
==============================================================================
DRS Tree at mohc_eg/
------------------------------------------------------------------------------
cmip5.output1.MOHC.HadGEM2-ES.rcp45.3hr.atmos.3hr.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.3hr.land.3hr.r1i1p1                 *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrLev.r1i1p1             *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrPlev.r1i1p1            *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.atmos.day.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.land.day.r1i1p1                 *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.aerosol.aero.r1i1p1.v20100927   -
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.atmos.Amon.r1i1p1               *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.land.Lmon.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.landIce.LImon.r1i1p1            *
==============================================================================
```

Using drs_tool's criteria options you can upgrade multiple datasets in one command:

```
$ drs_tool upgrade -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1 --realm=atmos --frequency=6hr
==============================================================================
DRS Tree at mohc_eg/
------------------------------------------------------------------------------
Upgrading cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrLev.r1i1p1 to version 20100927 ... done
Upgrading cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrPlev.r1i1p1 to version 20100927 ... done
==============================================================================
$ drs_tool list -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1
==============================================================================
DRS Tree at mohc_eg/
------------------------------------------------------------------------------
cmip5.output1.MOHC.HadGEM2-ES.rcp45.3hr.atmos.3hr.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.3hr.land.3hr.r1i1p1                 *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrLev.r1i1p1.v20100927   -
cmip5.output1.MOHC.HadGEM2-ES.rcp45.6hr.atmos.6hrPlev.r1i1p1.v20100927  -
cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.atmos.day.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.land.day.r1i1p1                 *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.aerosol.aero.r1i1p1.v20100927   -
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.atmos.Amon.r1i1p1               *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.land.Lmon.r1i1p1                *
cmip5.output1.MOHC.HadGEM2-ES.rcp45.mon.landIce.LImon.r1i1p1            *
==============================================================================
```

Finally you need to send publish the datasets with esgpublish. To make this easier drs_tool can create a mapfile of a dataset:

```
$ drs_tool mapfile -R mohc_eg/ -I mohc_eg/incoming/ cmip5.output1 --realm=aerosol >rcp45.mon.aerosol
$ head rcp45.mon.aerosol.map
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/loadsoa/loadsoa_aero_HadGEM2-
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/loadsoa/loadsoa_aero_HadGEM2-
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/loadsoa/loadsoa_aero_HadGEM2-
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/loadsoa/loadsoa_aero_HadGEM2-
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/loadbc/loadbc_aero_HadGEM2-ES
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/loadbc/loadbc_aero_HadGEM2-ES
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/loadbc/loadbc_aero_HadGEM2-ES
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/loadbc/loadbc_aero_HadGEM2-ES
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/wetbc/wetbc_aero_HadGEM2-ES_r
mohc_eg/output1/MOHC/HadGEM2-ES/rcp45/mon/aerosol/aero/r1i1p1/v20100927/wetbc/wetbc_aero_HadGEM2-ES_r
```

Some further examples of usage can be found in the doctest file test/test_command.txt.

1.4 CMIP5 product detection

drslib includes an algorithm for detecting the required CMIP5 product from filenames generated by CMOR2 developed by Martin Juckes. The module `drslib.p_cmip5` contains an API to the algorithm and `drs_tool` will use the algorithm to allocate a product for incoming data when used with the `--detect-product` option.

A detailed discussion of the algorithm used is available as the document Identifying the product [PDF].

1.4.1 `drs_tool` configuration

Product detection requires extra information to be configured about the CMIP5 experiment and the data being processed. Before first using the product detection feature you should initialise the CMIP5 experiment description data using `drs_tool init`:

```
$ drs_tool init --shelve-dir=DIR
```

where `DIR` is a directory suitable for containing the data files. On an ESG datanode a suitable command might be:

```
$ drs_tool init --shelve-dir=/usr/local/share/p_cmip5/data
```

In addition to the shelve directory `p_cmip5` requires additional information about the model output being processed to be included in an external configuration file (see **'Input configuration file'_**).

Both these parameters can be configured using metaconfig as follows:

```
[metaconfig]
configs = drslib

[drslib:p_cmip5]
shelve-dir = /usr/local/share/p_cmip5/data
config = /usr/local/share/p_cmip5/model.ini
```

1.4.2 The configuration file.

A configuration file is required by the p_cmip5 module (which assigns the data to DRS product=output1 or output2) for processing piControl data, and may optionally contain additional information which will ensure consistency of of the assigment for some other datasets (details below).

The configuration file is in standard ini-file format and should contain one section for each model name you want to process. Within each section set of option name/value pairs are listed to specify relevant properties of the model.

An Example

For instance the definition for 2 models `HADCM3` and `HIGEM1-2` could look as follows:

```
[HADCM3]

category=centennial
branch_year_piControl_to_historical=1820
base_year_historical=1850
branch_year_esmControl_to_esmHistorical=1850
base_year_esmHistorical=1850

[HIGEM1-2]
```

```
category=other
branch_year_piControl_to_historical=1820
base_year_historical=1850
branch_year_esmControl_to_esmHistorical=1850
base_year_esmHistorical=1850
```

Option names

1. category

Value either 'centennial' or 'other'

Description The category specifies which suite of experiments the model is being used for. This information is use to determine what data should be prioritised for quality control and DOI assignment. The aim is to ensure consistency between experiments and between modelling groups. If a model is used both for centennial and decadal experiments, specify 'centennial'.

2. branch_year_piControl_to_historical

Value integer

Description The year of the piControl data used to initiate the historical run.

Required if piControl data for tables aero, day or 6hrPlev is archived. This information can be determined from the global attribute "branch" in the historical data files and the base year from the time units of the piControl experiment.

3. base_year_historical

Value integer

Description The year of the start of the historical run. This is required becuase it is needed when processing piControl data, and thus cannot generally be obtained from the data files being processed.

4. branch_year_esmControl_to_esmHistorical

Value integer

Description The year of the piControl data used to initiate the historical run. See notes on 2. branch_year_piControl_to_historical

5. base_year_esmHistorical

Value integer

Description Start of esmHistorical expt. See notes on 3. base_year_historical

6. base_year_abrupt4xCO2 [optional]

Value integer

Description The year of start of the abrupt4xCO2 run. Used for processing abrupt4xCO2 data – only needed if the base year specified by the time units does not correspond to start of experiment.

7. base_year_piControl [optional]

Value integer

Description The year of start of the piControl run. Only needed if the base year specified by the time units does not correspond to start of experiment.

Used in determining which years of data from the aero table, piControl experiment in the decadal suite are replicated.

8. base_year_1pctCO2 [optional]

Value integer

Description The year of start of the 1pctCO2 run. Only needed if the base year specified by the time units does not correspond to start of experiment.

1.4.3 Invoking drs_tool with product detection

`drs_tool` will allocate a product to files in the incoming directory when invoked with the `--detect-product` option.

`drs_tool list` will show the product deduced in each dataset_id and list datasets for which the product could not be determined as incomplete. `drs_tool todo` and `drs_tool upgrade` will only operate on datasets for which the product could be determined.

In the following example data from the UK Met Office Hadley Centre is processed into the DRS hierarchy.

```
$ drs_tool list -I ./mohc_holding/ -R ./cmip5 --detect-product
...
[INFO] drslib.p_cmip5: Product deduced as output1, selected years [112/56] assigned to output1
[INFO] drslib.p_cmip5: Deducing product for <DRS cmip5.%.MOHC.HadGEM2-ES.historical.6hr.atmos.6hrPlev
[INFO] drslib.p_cmip5: Product deduced as output1, selected years [112/56] assigned to output1
[INFO] drslib.p_cmip5: Deducing product for <DRS cmip5.%.MOHC.HadGEM2-ES.historical.6hr.atmos.6hrPlev
[INFO] drslib.p_cmip5: Product deduced as output1, selected years [112/56] assigned to output1
[INFO] drslib.p_cmip5: Deducing product for <DRS cmip5.%.MOHC.HadGEM2-ES.historical.6hr.atmos.6hrPlev
[INFO] drslib.p_cmip5: Product deduced as output1, selected years [112/56] assigned to output1
[INFO] drslib.p_cmip5: Deducing product for <DRS cmip5.%.MOHC.HadGEM2-ES.historical.6hr.atmos.6hrPlev
[INFO] drslib.p_cmip5: Product deduced as output1, selected years [112/56] assigned to output1
==============================================================================
DRS Tree at .
------------------------------------------------------------------------------
cmip5.output1.MOHC.HadGEM2-ES.historical.6hr.atmos.6hrLev.r1i1p1        0:0 1120:1371721676416
cmip5.output1.MOHC.HadGEM2-ES.historical.6hr.atmos.6hrPlev.r1i1p1       0:0 224:116004074368
------------------------------------------------------------------------------
2 datasets awaiting upgrade
==============================================================================

# Select the 6hrPlev dataset for publishing and check commands to be issued
```

```
$ drs_tool todo -I ./mohc_holding/ -R . --detect-product cmip5.%.%.%.%.%.%.6hrPlev
================================================================================
DRS Tree at .
--------------------------------------------------------------------------------
Publisher Tree cmip5.output1.MOHC.HadGEM2-ES.historical.6hr.atmos.6hrPlev.r1i1p1 todo for version 201

mv ./mohc_holding/psl_6hrPlev_HadGEM2-ES_historical_r1i1p1_194912010600-195012010000.nc ./cmip5/outpu
ln -s ./cmip5/output1/MOHC/HadGEM2-ES/historical/6hr/atmos/6hrPlev/r1i1p1/files/psl_20101006/psl_6hrP
...
mv ./mohc_holding/va_6hrPlev_HadGEM2-ES_historical_r1i1p1_200412010600-200512010000.nc ./cmip5/output
ln -s ./cmip5/output1/MOHC/HadGEM2-ES/historical/6hr/atmos/6hrPlev/r1i1p1/files/va_20101006/va_6hrPle
================================================================================

# Do the upgrade
$ drs_tool upgrade -I ./mohc_holding/ -R . --detect-product cmip5.%.%.%.%.%.%.6hrPlev
...

# List the results.
# Not including --detect-product lists datasets that are incompletely specified
$ drs_tool list -I ./mohc_holding -R mohc_dryrun
================================================================================
DRS Tree at mohc_dryrun
--------------------------------------------------------------------------------
cmip5.output1.MOHC.HadGEM2-ES.historical.6hr.atmos.6hrPlev.r1i1p1.v20101005  224:116004074368
--------------------------------------------------------------------------------
Incompletely specified incoming datasets
--------------------------------------------------------------------------------
cmip5.%.MOHC.HadGEM2-ES.historical.6hr.atmos.6hrLev.r1i1p1
================================================================================
```

1.5 DRS Schemes

In order to support multiple projects beyond CMIP5, DRSlib has been extended to support multiple DRS schemes. The `--scheme` switch to `drs_tool` will allow you to select form a set of configured schemes, currently:

1. CMIP5 (default)
2. SPECS
3. CORDEX

Each scheme has a different set of DRS components and component ordering.

When operating on projects other than CMIP5 drslib is designed to perform less metadata checks. These functions are now performed by drslib's sister tool ceda-cc. The `ceda_cc` tool will verify files meet a project's metadata requirements and output a JSON file which can be used by drslib to deduce certain metadata values that are not apparent from the files. This JSON file is passed to `drs_tool` using the *–json-drs* option.

CHAPTER 2

Reference

2.1 Module Reference

2.1.1 `drslib.drs` – DRS objects and utilities

The drs module contains a minimal model class for DRS information and some utility functions for converting filesystem paths to and from DRS objects.

More sophisticated conversions can be done with the `drslib.translate` and `drslib.cmip5` modules.

class `drslib.drs.`**`BaseDRS`**(**argv*, ***kwargs*)

Base class of classes representing DRS entries.

This class provides an interface to: 1. Define and expose the components of the DRS and their order 2. Convert components in and out of serialised form 3. Determine whether a DRS entry is complete 4. Define the publishing level of datasets represented by this DRS

This class provides default implementations of: 1. serialisation to dataset-id with or without version

Subclasses decide what components make up the DRS.

Variables

- **DRS_ATTRS** – a sequence of component names in the order they appear in the DRS identifier.
- **PUBLISH_LEVEL** – the last component name which is part of the published dataset-id.

classmethod **`from_dataset_id`**(*klass*, *dataset_id*, ***components*)

Return a DRS object fro a ESG Publisher dataset_id.

If the dataset_id contains less than 10 components all trailing components are set to None. Any component of value '%' is set to None

E.g. >>> drs = DRS.from_dataset_id('cmip5.output.MOHC.%.rpc45') >>> drs.institute, drs.model, drs.experiment, drs.realm ('MOHC', None, 'rpc45', None)

classmethod **`from_json`**(*klass*, *json_obj*, ***components*)

Create a DRS object from a ceda-cc compatible json object.

ceda-cc may use different keys than the DRS terms used internally so these should be mapped here.

Json_obj A dictionary containing a ceda-cc representation of the drs terms as exported in json.

`is_complete`()

Returns boolean to indicate if all components are specified.

Returns `True` if all components excluding those in self.OPTIONAL_ATTRS have a value.

is_publish_level()
Returns boolian to indicate if the all publish-level components are specified.

to_dataset_id(*with_version=False*)
Return the esgpublish dataset_id for this drs object.

If version is not None and with_version=True the version is included.

class drslib.drs.**CmipDRS**(**argv*, ***kwargs*)
Represents a DRS entry. DRS objects are dictionaries where DRS components are also exposed as attributes. Therefore you can get/set DRS components using dictionary or attribute notation.

In combination with the translator machinary, this class maintains consistency between the path and filename portion of the DRS.

Variables

- **activity** – string
- **product** – string
- **institute** – string
- **model** – string
- **experiment** – string
- **frequency** – string
- **realm** – string
- **variable** – string
- **table** – string of None
- **ensemble** – (r, i, p)
- **version** – integer
- **subset** – (N1, N2, clim) where N1 and N2 are (y, m, d, h, mn, sec) and clim is boolean
- **extended** – A string containing miscellaneous stuff. Useful for representing irregular CMIP3 files

class drslib.drs.**DRSFileSystem**(*drs_root*)
Represents the mapping scheme between `DRS` objects and a filesystem.

Instances of this class deal with how DRS objects are partitioned into Publicaiton-level datasets and how files within a DRS are mapped to the filesystem.

Variables

- **drs_cls** – The subclass of `BaseDRS` used in this filesystem.
- **publish_level** – the last component name which is part of the published dataset-id.
- **drs_root** – The path to the root directory of a DRS filesystem. This path represents the activity level of the DRS.

drs_to_linkpath(*drs*, *version=None*)
Return the full path of the symbolic link for this drs

drs_to_publication_path(*drs*)
Returns a directory path from a `DRS` object. Any DRS component that is set to None will result in a wildcard '*' element in the path.

This function does not take into account of MIP tables of filenames.

Parameters **drs** – The `DRS` object from which to generate the path

drs_to_realpath(*drs*)
Return the full path to the real file for *drs* (as oposed to the symbolic link).

drs_to_storage(*drs*)
Return the subpath within the files directory for this `DRS` instance.

filename_to_drs(*filename*)
Return a DRS instance deduced from a filename.

filepath_to_drs(*filepath*)
Return a DRS instance deduced from a full path.

iter_files_with_links(*pub_dir*, *version=None*, *into_version=None*)
Iterate over files of a particular version also returning it's respective link into the latest version.

Parameters

- **version** – iterate over a specific version or all versions if None
- **into_version** – the version into which symbolic links will be made, if None same as version, if both are None same as self.latest

Yield filepath, linkpath

publication_path_to_drs(*path*, *activity=None*)
Create a `DRS` object from a filesystem path.

This function is more lightweight than using `drslib.translator` but only works for the parts of the DRS explicitly represented in a path.

Parameters **path** – The path to convert. This is either an absolute path

or is relative to the current working directory.

storage_to_drs(*subpath*)
Return a `DRS` instance representing the DRS components deducible from its subpath within the files directory.

2.1.2 drslib.translate – Translate DRS filepaths

2.1.3 drslib.cmip5 – DRS objects and utilities

2.1.4 drslib.mip_table – Loading CMOR MIP Tables

Simple parser for MIP tables.

My interpretation of the format from reading the CMIP5 tables.

class drslib.mip_table.**MIPTable**(*filename*)
Hold information from a MIP table.

This information is used to enforce DRS vocabularies.

Property name The name of the MIP table as used in DRS filenames.

Property variables A list of variables in this table.

Property experiments A list of valid experiment ids for this table.

get_variable_attr(*variable*, *attr*)
Retrieve an attribute of variable.

If the attributes isn't in the variable entry the global value is returned

class drslib.mip_table.**MIPTableStore**(*table_glob*)
Holds a collection of mip tables.

Property tables A mapping of table names to IMIPTable instances

add_table(*filename*)
Read filename as a MIP table and add it to the store.

Returns The added MIPTable instance.

get_global_attr(*table*, *attr*)
Return global table attribute.

get_global_attr_mv(*table*, *attr*)
Return the value of a variable's attribute in a given table.

get_variable_attr(*table*, *variable*, *attr*)
Return the value of a variable's attribute in a given table.

get_variable_attr_mv(*table*, *variable*, *attr*)
Return the value of a variable's attribute in a given table.

drslib.mip_table.**iter_entries**(*fh*)
Generate events (entry_name, value_dict) by reading a MIP table from a file object.

drslib.mip_table.**iter_table**(*fh*)
Generates events (entry, value, comment) by reading a MIP table from a file object.

drslib.mip_table.**read_model_table**(*table_csv*)
Read Karl's CMIP5_models.xls file in CSV export format and return a map of institute to model name.

This function is invoked internally to load CMIP5_models.xls from inside drslib.

drslib.mip_table.**split_comment**(*line*)
Detect comment.

Quoted '!' characters are detected.

2.1.5 drslib.drs_tree – Managing DRS directory structure versioning

2.1.6 drslib.p_cmip5 – CMIP5 product detection

The p_cmip5 module will decide whether data should be assigned to DRS product=output1 or output2. Background and the algorithm steps are given in requested_subset_decision_tree_v0_5.pdf.

Set-up

There is a set-up step performed by the code in p_cmip5/init.py, using the "init(shelve_dir)" function. This modules takes information from the spreadsheets CMIP5_archive_size_template.xls and standard_output_17Sep2010_mod.xls and stores it in python shelves used by the main code. The shelves are placed in "shelve_dir".

The cmip5_product class

The `drslib.p_cmip5.product` module provides a `cmip5_product` class with a "find_product" method. At instantiation, the class picks up configuration tables The configuration file (sample in ini/sample_1.ini, ini/smaple_2.ini) needs to contain information about each model. This information is used for a small number of cases and the format is not yet stable:

```
class cmip5_product:

  def __init__(self,mip_table_shelve='sh/standard_output_mip',
                    template='sh/template',
                    stdo='sh/standard_output',
                    config='ini/sample_1.ini',
                    override_product_change_warning=False,
                    policy_opt1='all_rel',not_ok_excpt=False):
```

Optional arguments

mip_table_shelve shelve containing information about MIP tables;

template shelve containing information mapping experiment names to labels used in the standard_output spreadsheet;

stdo shelve containing information from the standard_output spreadsheet;

config configuration file;

override_product_change_warning in some cases it is possible that adding new data to previously published data can change the product designation of the previously published data. The default behaviour is to givw an error return at this point. This can be overridden, the code will then provide the product of the file to be added and lists of changes that need to be made to previously published data. It is not expected, however, that the ESG publisher will support such updates: the user should instead compile a new set of files to submit using all the previously published files and the new files.

policy_opt1 this controls two options for treatment of data blocks in which time slices are requested. The default is that, if the time slices are specified using relative dates (e.g. relative to start of experiment) and the number of years submitted is les than the number of years requested, all years submitted will be assigned to output1 without examining the dates. There is an option (depricated) to extend this catch-all approach to time slices specified with absolute dates.

not_ok_excpt if True, raise an exception if product can not be designated as output1 or output2.

The find_product method

The principal interface to the cmip5_product class is the find_product method:

```
def find_product(self,var,table,expt,model,path,startyear=None,endyear=None,verbose=False,
                 path_output1=None, path_output2=None,selective_ads_scan=True):
```

Required arguments

var DRS variable name

table MIP table

expt DRS experiment name

model Model name

path Path to directory containing all the files of one atomic dataset.

Optional arguments

startyear first year of the file to be assessed

endyear last year of file to be assessed (not currently used)

verbose if True, provide additional comments to logger

path_output1 path to last published output1 data for this atomic dataset, if new data is to be considered as an addition;

path_output2 path to last published output1 data for this atomic dataset, if new data is to be considered as an addition;

selective_ads_scan when scanning the atomic dataset directory, look only at files matching the variable, table, experiment and model. The False option is provided to facilitate testing using dummy data files, and should not otherwise be used.

return if "not_ok_excp=True", the method will return True if it can assign output1 or output2, otherwise an exception will be raised. if "not_ok_excpt=False", the method will return False when it cannot assign output1 or output2, with a message in pc.reason (where pc is an instance of the cmip5_product class).

Attributes containing results

After successful completion of the find_product method, the followin attibutes of the instance contain information:

product the product to be assigned to the file;

reason a short summary of the reasons for the assigment;

rc a return code, 'OKnnn' if successful, 'ERRnnn' if not.

Usage

The following code fragment illustrates usage of the module:

```
## import module
import p_cmip5_v5 as p

## create and instance of the cmip5_product class, specifying a configuration file
##
pc2 = p.cmip5_product( config='ini/sample_2.ini')

## test a file: using the variable, mip table, experiment id, model and specifying the path of the a
## the submitted files.
## In some cases the decision as to which product the file belongs in will depend on the contents of
## verbose=True results in additional messages being printed to standard out.

if pc2.find_product( var, mip, expt,model,path,startyear=startyear, verbose=verbose):
  print 'product is: ', pc2.product
```

```
##  A True return means the method has identified the product
else:
##  A False return means the product could not be identified
  print 'Dont know what to do with this data:: ',pc2.reason
```

Testing the p_cmip5 module: test_p_cmip5.py

The test_p_cmip5 module can be used to test the p_cmip5 module. E.g. run the following from the directory containing the "test" subdirectory:

```
$ nosetests --tests=test/test_p_cmip5.py
```

CHAPTER 3

Appendix

3.1 Translating CMIP3 to CMIP5

The script `translate_cmip3` converts the CMIP3 archive into a form as close to the DRS specification as possible. This transformation involves both filename and directory structure changes. From the command's help message:

```
Usage: translate_cmip3 [options] cmip3_root cmip5_root

Options:
  -h, --help            show this help message and exit
  -i INCLUDE, --include=INCLUDE
                        Include paths matching INCLUDE regular expression
  -e EXCLUDE, --exclude=EXCLUDE
                        Exclude paths matching EXCLUDE regular expression
  -c, --copy            Copy rather than move files
  -d, --dryrun          Emit log messages but don't translate anything
  -l LOGLEVEL, --loglevel=LOGLEVEL
                        Set logging level
```

3.1.1 Example

The drslib.cmip3 module implements a similar API to drslib.cmip5 thus allowing CMIP3 paths to be converted to DRS instances then converted into CMIP5 DRS format.

```
>>> from drslib import cmip3
>>> cmip3_trans = cmip3.make_translator('cmip3')
>>> drs3 = cmip3_trans.filepath_to_drs('cmip3/20c3m/atm/da/rsus/gfdl_cm2_0/run1/rsus_A2.19610101-1965
>>> drs3
<DRS activity='cmip3', product='output', institute='GFDL', model='CM2', experiment='20c3m', frequency
>>> cmip5_trans.drs_to_filepath(drs3)
'http://example.com/cmip5/output/GFDL/CM2/20c3m/day/atmos/v1/rsus/r1/rsus_A2_CM2_20c3m_r1_19610101-19
```

3.1.2 CMIP3 DRS components

The CMIP3 activity is `cmip3` and all datasets are given the product `output`. The version component is always `v1`. Translation of the other DRS components for CMIP3 are described below.

Institute & Model

Institutes and models given in capital letters and underscores are converted to dash characters. Capitalisation is chosen to be consistent with the examples given in sections 3.2 and 3.3 of the DRS specification and dashes are used to avoid ambiguity in DRS filenames that use underscores as the component separator.

Where the exact encoding is not trivial the syntax used by the IPCC Data Distribution Centre [DDC] is used.

CMIP3 directory	Institute	Model
bcc_cm1	CMA	BCC-CM1
bccr_bcm2_0	BCCR	BCM2
cccma_cgcm3_1	CCCMA	CGCM3-1-T47
cccma_cgcm3_1_t63	CCCMA	GCM3-1-T63
cnrm_cm3	CNRM	M3
miub_echo_g	MIUB-KMA	CHO-G
csiro_mk3_0	CSIRO	K3
csiro_mk3_5	CSIRO	K3-5
gfdl_cm2_0	GFDL	M2
gfdl_cm2_1	GFDL	M2-1
inmcm3_0	INM	M3
ipsl_cm4	IPSL	M4
iap_fgoals1_0_g	LASG	GOALS-G1-0
mpi_echam5	MPIM	CHAM5
mri_cgcm2_3_2a	MRI	GCM2-3-2
giss_aom	NASA	ISS-AOM
giss_model_e_h	NASA	ISS-EH
giss_model_e_r	NASA	ISS-ER
ncar_ccsm3_0	NCAR	CSM3
ncar_pcm1	NCAR	CM
miroc3_2_hires	NIES	IROC3-2-HI
miroc3_2_medres	NIES	IROC3-2-MED
ukmo_hadcm3	UKMO	ADCM3
ukmo_hadgem1	UKMO	ADGEM1
ingv_echam4	INGV	CHAM4

Experiment

The experiment component remains unchanged from the CMIP3 archive structure except that it's position in the tree changes to match the DRS specification.

Frequency

The CMIP3 frequency specifiers are translated into those described in the DRS specification as follows:

CMIP3	DRS
yr	yr
mo	mon
da	day
3h	3hr
fixed	fx

Modelling-realm

We map CMIP3 realms onto equivilent CMIP5 realms. In some cases this mapping also depends on the variable. This mapping is defined in the table below:

CMIP3 realm	Variable	DRS realm
atm	mrsos	land
atm	trsult	aerosol
atm	trsul	aerosol
atm	tro3	atmosChem
atm	*	atmos
ice	*	seaIce
land	sftgif	landIce
land	*	land
ocn	*	ocean

Variable name

Variable names are left unchanged.

Ensemble member

The encoding `run<N>` is translated into `r<N>`.

Subset and extended path

Although most filenames in the CMIP3 archive follow a consistent syntax there are enough exceptions to make complete adherence to the DRS specification impractical. Instead `translate_cmip3` attempts to extract the variable, MIP table name from the CMIP3 path and constructs an approximate DRS filename of the form:

```
<variable>_<mip-table>_<model>_<experiment>_<ensemble-member>_<extended>.nc
```

where `<extended>` is the unparsed portion of the filename that may contain a temporal subset or may be irregular. Some examples are given below::

```
/20c3m/atm/da/rsus/gfdl_cm2_0/run1/rsus_A2.19610101-19651231.nc --> rsus_A2_CM2_20c3m_r1_19610101-196
/1pctto2x/atm/mo/rlftoaa_co2/ipsl_cm4/run1/rlftoaa_co2_A5_1860-1869.nc --> rlftoaa_co2_A5_CM4_1pctto2
/2xco2/land/fixed/orog/miroc3_2_hires/run1/orog_A1.nc --> orog_A1_MIROC3-2-HI_2xco2_r1.nc
/sresa1b/atm/mo/rlut/cccma_cgcm3_1/run4/rlut_a1_sresa1b_4_cgcm3.1_t47_2001_2100.nc --> rlut_a1_CGCM3-
```

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

[DDC] http://www.ipcc-data.org

d