

Additional CDAT features:
***cdscan* – for file aggregation**

The “cdscan” utility (1)

- A very valuable feature of CDAT is its file aggregation utility **cdscan**.
- This allows you to describe an entire dataset with just one XML file, that is opened by CDAT using the standard **cdms.open()** call.
- The XML format is known as *Climate Data Markup Language (CDML)* which is fully described in the CDAT manual.
- **cdscan** removes the need to know about filenames by providing a global description of a collection of files. Metadata and aggregation are handled together.

The “cdscan” utility (2)

```
/usr/local/cdat/bin/cdscan -i 6 -d "ERA-40"  
-x e40-1.0-as-1958-2001.xml  
/era40/data/gp/as/**/*/*.ctl
```

- The resulting CDML file (e40-1.0-as-1958-2001.xml) contains the following sections:
 - <dataset> - general information at the dataset level.
 - <axis> - axis dimension information.
 - <variable> - relating to individual variables.
- ***It describes and links to over 3,000,000 files and is only 21KB in size!***

The “cdscan” utility (3)

- **cdscan** will look through your archive analysing all the files you point it to, extracting variable information, axis information, global metadata and grouping any commonalities it finds.
- Let's have a look at it in action:
 - We have an archive of 1200 monthly mean NetCDF files in the current directory that range from 1901 to 2000. We would like an application to access them all without needing to know about individual file names.
 - **Scenario 1**: Filenames do not map nicely to their contents. So we run `cdscan plain` and see what comes out.

```
cdscan -x monthly_means.xml ./*.nc
```

Using templates for filenames

- **Scenario 2:** Filenames reflect the contents of the files closely with the file-naming convention:

`<YYYY><MM>_<VARIABLE>.nc`

- In the olden days, cdsctl used to be “**cdimport**” which had one excellent feature you might want to make use of. It allows you to add a template for file and directory names.
- The template allows you to specify time components, start and end levels as well as variable IDs.

“cdimport”: a former life

`$ cdimport -h # yields information about the template:`

```
%d day number (1 .. 31)
%eX ending timepoint/level, where X is a specifier character
%f day, two-digit, zero-filled (01, 02,..., 31)
%g month, lower case, three characters ('jan', 'feb', ...)
%G month, upper case, three characters ('JAN', 'FEB', ...)
%H hour (0 .. 23)
%h hour, two-digit, zero filled (00, 01, ..., 23)
%L vertical level (integer)
%m month number, not zero filled (1 .. 12)
%M minute 0 .. 59
%n month number, two-digit, zero-filled (01, 02, ..., 12)
%S second (0 .. 59)
%v variable ID (string)
%y year, two-digit, zero-filled (integer)
%Y year (integer)
%z Zulu time (ex: '6Z19990201')
%% percent sign
```

Template parameters in cdmsobj.py

- The best place to look for a complete list of templates is in the **cdmsobj** module located at:
`<your_cdat>/lib/python23/site-packages/cdms/cdmsobj.py`
- This module lists all the patterns that CDAT can translate when reading data from a CDML file using a template.

Back to the example

- **Scenario 2:** Filenames reflect the contents of the files closely with the file-naming convention:

```
<YYYY><MM>_<VARIABLE>.nc
```

- Run `cdscan` with the `-p` argument and your template:

```
cdscan -x monthly_means.xml  
-p %Y%n_%v.nc /*.nc
```

- Optionally, you can do a manual edit of the XML file to tidy up the unused `<cdms_filemap>` attribute:
 - Remove the line containing `<cdms_filemap>` which may hold up to millions of elements if you have a lot of files (this is time-consuming when parsing).

What else can cdscan do? (1)

- Let's look at the help output from "cdscan -h":
 - a **alias_file**: change variable names to the aliases defined in an alias file.
 - c **calendar**: either "gregorian", "proleptic_gregorian", "julian", "noleap", or "360_day". Default:
 - d **dataset_id**: dataset identifier. Default: "none"
 - e **newattr**: Add or modify attributes of a file, variable, or axis.

What else can cdscan do? (2)

- exclude var,var,...:** exclude listed variables from output.
- f file_list:** file containing a list of absolute data file names, one per line.
- h:** print a help message.
- i time_delta:** scan time as a 'linear' dimension. This is useful if the time dimension is very long.
- include var,var,...:** only include the listed variables in the output.

What else can cdscan do? (3)

- j:** scan time as a vector dimension. Time values are listed individually. Turns off the -i option.
- l levels:** list of levels, comma-separated. Only specify if files are partitioned by levels.
- m levelid:** name of the vertical level dimension. The default is the name of the vertical level dimension.
- p template:** Compatibility with pre-V3.0 datasets. 'cdimport -h' describes template strings.
- q:** quiet mode

What else can cdscan do? (4)

- r **time_units**: time units of the form "<units> since yyyy-mm-dd hh:mi:ss", where <units> is one of "year", "month", "day", "hour", "minute", "second".
- s **suffix_file**: Append a suffix to variable names, depending on the directory the data is located in, deals with multiple files holding variables with the same name.

What else can cdscan do? (5)

- t timeid:** id of the partitioned time dimension. The default is the name of the time dimension.
- time-linear tzero,delta,units[,calendar]:** Override the time dimension(s) with a linear time dimension. The arguments are a comma-separated list.
- x xmlfile:** XML filename. By default, output is written to standard output.

So what does the user see?

- **cdscanned** files are same as any other CDAT-compatible data file:

```
>>> import cdms
>>> f=cdms.open('cdscanned_stuff.xml')
>>> print f.variables # Will list the
    variables
>>> var=f('q', time=("1910-10", "1940-09"),
    lat=(30,60), lon=(-20,10), level=1000)
# var now holds the contents of whatever
    actual data files needed to be aggregated
    together.
```

- As a user you see none of this and can get on with your science!

So why use cdscan?

Some reasons for using cdscan:

1. It allows large datasets to be described as a grouped entity.
2. It removes the requirement of data format knowledge.
3. It removes the requirement of file-name knowledge.
4. Datasets can be sliced in any way the user chooses using logical spatiotemporal selectors rather than loops of programming code.
5. You can use it to improve the metadata of your data files...

cdscan to up your metadata quality!

- Since cdscan exposes a common set of metadata for a dataset it can be used to *improve your CF-compliance!*
- Use the '-e' argument to add new attributes to your variables, axes and at the global file level:

```
-e temp.standard_name="air_temperature"  
-e temp.units="K"  
-e level.standard_name="depth"  
-e .source="UK Met Office Unified Model Version 5.5"  
-e .references="Cited in paper by E.S.Fuller (2001)."
```

Additional CDAT features:
***cdtime* – time utilities**
module

The “*cdtime*” module

- CDAT uses its own *cdtime* module to manage time objects and temporal coordinate reference systems (or calendars).
- Whilst *cdtime* is integral to the internal workings of the CDMS package, it is also a very useful tool for users interacting with data.
- *cdtime* provides useful functionality to:
 - create time objects attached to datasets.
 - refer to different calendar types (such as Gregorian and 360-day year calendars).
 - convert between an absolute and relative time description.
 - work with time intervals.

cdtime docs: http://esg.llnl.gov/cdat/cdms_html/cdms-3.htm

Component and Relative Time

- Importing: `import cdtime`
- Two time types included:
 1. Component Time:
 - Integer fields for (year, month, day...second)
 - **e.g.** `ct=cdtime.comptime(1999, 6, 12, 18)`
for 12/06/1999 18:00
 2. Relative Time:
 - A floating point value and a relative units string (and base time).
 - **e.g.** `rt=cdtime.relttime(11.75, "days since 1999-06-01 00:00")` # for 12/06/1999 18:00
- The two types provide helpful functionality and versatility when representing time objects.
- Both functions return a *cdtime object*. This then has a number of methods available.

The standard time units definition

- But first, let's get familiar with NetCDF-style time definitions...
- cftime and CDMS time axes follows the NetCDF convention for representing time. Relative time is time relative to a fixed base time. It consists of:
 - a **units** string, of the form “**units since basetime**”, and a floating-point **value**.

Time units from UDUNITS

The specification [from UDUNITS]:

“seconds since 1992-10-8 15:15:42.5 -6:00”

- indicates seconds since October 8th, 1992 at 3 hours, 15 minutes and 42.5 seconds in the afternoon in the time zone which is six hours to the west of Coordinated Universal Time (i.e. Mountain Daylight Time).
- The time zone specification can also be written without a colon using one or two-digits (indicating hours) or three or four digits (indicating hours and minutes).

Using cftime

So how do you use cftime? Here are some examples:

1. You know a time interval and the units but no absolute (component) time:

```
ct=cftime.comptime(10, "hours since 1981-1-1")
```

2. You have a dataset with a known basetime (reference time) and you are adding a new value with a known absolute (component) time:

```
# mytime is already a component time  
reltime=mytime.torel("days since 1996-1-1")
```

3. You would like to add 365 days to a cftime object:

```
newtime=ct.add(365, cftime.Days)
```

4. To compare two time objects:

```
if ct1.cmp(ct2): print "They are the same!"
```

Calendars in cftime

A calendar specifies the number of days in each month, for a given year. cftime supports the following:

- Gregorian
- Mixed (Julian/Gregorian) = Default
- Julian
- NoLeap (Year)
- Calendar360 (all days have 30 months).

Typical usage is when you are converting or modifying cftime objects:

```
newtime=ct1.add(25, cftime.Hours, calendar=cftime.Calendar360)  
newtime2=reft.tocomp(calendar=cftime.GregorianCalendar)
```