The PRISM Data Processing and Visualisation System

Edited by:
Steve Mullerworth and the Processing and Visualisation Team

PRISM-Report Series-15

1st Edition
(last change: December 17, 2004)
How to get assistance?

The individual work packages of the PRISM project can be contacted as listed below. PRISM publications can be downloaded from the WWW server of the PRISM project under the URL: <http://prism.enes.org/Results/Documents/>

Phone Numbers and Electronic Mail Addresses

Electronic mail addresses of the individual work packages are composed as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
<th>PRISM Work Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steve Mullerworth</td>
<td>+44-1392-886899</td>
<td>wp4a</td>
</tr>
<tr>
<td>The Processing and Visualisation Team</td>
<td></td>
<td>wp4a</td>
</tr>
</tbody>
</table>
# Contents

1 Data Processing and Visualization ................................. 1
   1.1 Introduction .................................................. 1
   1.2 Review of Visualization Packages .............................. 2
   1.3 Processing with COCO ......................................... 3
      1.3.1 Detailed documentation of COCO methods ................. 4
      1.3.2 Tutorials and Examples .................................. 7
      1.3.3 Links to reference materials .............................. 8
      1.3.4 Developer Documentation ................................. 8
      1.3.5 Design notes and diagrams ............................... 10
      1.3.6 Notes for developers ...................................... 11
   1.4 Accessing Remote Climate Data ................................. 11
   1.5 Data Visualization with OpenDX ............................... 11
      1.5.1 Introduction .............................................. 11
      1.5.2 NetCDF CF-1.0 Import Modules ............................ 12
      1.5.3 Examples .................................................. 17
      1.5.4 References ............................................... 18
      1.5.5 Installing OpenDX and NetCDF CF-1.0 Modules ............. 23
      1.5.6 Limitations ............................................... 24
      1.5.7 Future Developments ..................................... 24
   1.6 Low End Visualization ......................................... 24
      1.6.1 Introduction .............................................. 24
      1.6.2 CDAT and VCS ............................................. 26
      1.6.3 Python ................................................... 27
      1.6.4 The ParaGen Code Generator .............................. 27
      1.6.5 The XML colour editor ................................. 29
      1.6.6 Low End Graphic Package ............................... 30
      1.6.7 Generation of Overviews (Thumbnails and html pages) .... 34
   1.7 VTK .......................................................... 36
   1.8 The VTK_Mapper Application .................................... 36
      1.8.1 Software architecture .................................... 36
      1.8.2 Covered features ......................................... 36
      1.8.3 Examples of use .......................................... 38
      1.8.4 Usage documentation ..................................... 43
      1.8.5 Installation .............................................. 44
      1.8.6 Conclusion and Future Work .............................. 44
      1.8.7 Useful On-line References ............................... 45
   1.9 Summary ...................................................... 45
   1.10 Acknowledgements .............................................. 46
List of Figures

1.1 COCO Architecture ............................................................. 10
1.2 A simple OpenDX network .................................................. 19
1.3 A OpenDX network applying a custom colormap ...................... 20
1.4 A simple OpenDX network .................................................. 21
1.5 Advanced visualization ...................................................... 22
1.6 Low end versus high end .................................................... 25
1.7 The ParaGen interface generator .......................................... 28
1.8 The colour table editor ...................................................... 30
1.9 An example of an irregular grid, plotted by LE parameter. ............ 32
1.10 Thumbnail layout ............................................................... 34
1.11 Different model grids with VTK_Mapper .............................. 39
1.12 Different representation modes in VTK_Mapper ....................... 40
1.13 Isocontour in a linear projection with VTK_Mapper .................. 40
1.14 Switching between representation modes with VTK_Mapper ......... 41
1.15 The Qt user interface .......................................................... 42
1.16 Off screen calling with VTK_Mapper ................................. 42
List of Tables

1.1 PRISM specific OpenDX modules appearing in the Import and Export category. . . . . . . 12
Chapter 1

Data Processing and Visualization

1.1 Introduction

This chapter describes the processing and visualization tools that have been developed to handle model data in the agreed PRISM data format, NetCDF(CF). Tools have been adapted or developed to handle this format. This chapter discusses the tools and packages chosen or developed for PRISM. It covers the following four main areas:

- Tools for processing data in the PRISM data format.
- Methods for archiving data and for accessing archived data.
- Tools to visualize PRISM data.
- Systems for automatically generating plots of model data. For example, a system that generates periodic plots of the output of a PRISM coupled climate simulation that can be viewed using a web browser to enable a scientist to check the progress of the run.

The PRISM data processing library is COCO (CDMS overloaded for CF Objects). It is written in Python and is based on the Climate Data Management System (CDMS). CDMS is, in turn, implemented as part of the Climate Data Analysis Tool (CDAT).

Visualization is divided into high end and low end visualization within PRISM. High end visualization indicates that the visualization tools and the data being visualized are available to the user on the local computer system. Direct access to the tools and data allows more flexibility and interactivity for the user when generating and manipulating plots: plots can be combined, images can be rotated, animations can be generated and projections can be changed all in real time.

Low end visualization refers to the automatic plotting systems. Plots are generated on a remote machine as and when the model outputs data. Choice of plot for the general user is limited to what plotting options have been made available on the remote machine and what has been selected by the user when configuring the run. However, apart from the reduced flexibility, the plots need to be of as high quality as those produced by the high end systems.

The OpenDX tool is the chosen high end visualization package. OpenDX incorporates a visual program editor (VPE) that allows users to visualize climate data and to directly manipulate the graphical output. The VPE includes a graphical user interface builder (GUI) so that OpenDX can also be used by software developers to build visualization applications for their users. A range of specially written modules enables the OpenDX tool to read in NetCDF(CF) data files. A number of example applications have been developed to demonstrate the abilities of OpenDX.

The low end system required a number of subsystems: a visualization package, a data processing package, example scripts to implement well defined processing and visualization operations, a user interfaces to to allow the selection of plotting options and a method of triggering the low end scripts to generate the plots at the appropriate time. A different visualization package from the high end called VCS was chosen to
generate the plots because it has a scripting interface which makes it easier to develop scripts that combine processing and visualization operations. It is intended that options within the PRISM user interface will allow the user to select variables and specify plots to be generated. The scripts are triggered by the PRISM run environment when data is produced by the model. Plots can be viewed using a web browser.

A third visualization tool called VTK has been explored. VTK offers the ability to develop high-quality high end applications and also has a Python scripting interface that means it aligns well with COCO processing and with the low end visualization system. VTK is a 2D/3D computer graphics, image processing, and visualization library widely used by visualization expert groups and ready to create a reusable infrastructure for parallel and distributed visualization components which can be used to solve terascale visualization problems (further information at http://www.vtk.org/). To demonstrate this and be sure that the VTK would fit well in the PRISM visualization and analysis infrastructure, a prototype application has been developed keeping in mind features such as a vector-based quality prints, 2D and 3D rendering, batch and off screen running mode.

Two novel tools have been created to assist users of the PRISM visualization and processing systems. The first is a tool called ParaGen which defines an XML schema for processing and visualization scripts which incorporates descriptions of their interfaces and their documentation. Given such an XML script description, ParaGen can generate instances of the script to be used in different environments. The benefits of this are that the same script can be run from a UNIX command line, or perhaps can be run from within a Python, C or Fortran program. ParaGen can also generate the CGI program that allows the user to control the script through a web page, or, relevant to visualizing the output from PRISM model runs, it can extract the information required to generate an input panel in the PRISM user interface.

The second tool is a colour table tool which provides users with a common way of specifying colour tables. Once a user has specified colour choices, the tool can generate colour tables in the required format for each of the PRISM visualization tools. The aim is that, for example, a plot generated automatically by the low end system can be regenerated in some form using the high end tools. For example, a series of plots generated using the low end tool could be regenerated into an interactive animation using the high end tool, the common aspect being that the colours remain the same or similar.

### 1.2 Review of Visualization Packages

A number of visualization packages were reviewed prior to selection of the packages for PRISM: AVS, GrADS, Ferret, IDL, NCAR Graphics, OpenDX, VisAD, VIS5D and VTK. The selection was based on the results of a users’ questionnaire and a technical review that together considered the perceived requirements, the technical issues involved with using each package (in particular with regard to using NetCDF(CF), the agreed PRISM data format), the quality of underlying support that each package already had, licensing issues, and the experiences of the development team in using each package.

In summary, AVS and IDL were rejected as they are not open source tools and thus do not accord with the agreed PRISM philosophy. Evaluation found that NCAR Graphics was not sufficiently user friendly. GrADS, VisAD and VIS5D did not support irregular grids which was one of the essential user requirements. In the end, the remaining tools, OpenDX and VTK, were both found to be suitable packages that would meet the requirements. Both of these tools are general visualization tools rather than being developed specifically for climate visualization, and they both enjoy wide usage in many other fields. General-purpose tools typically are more powerful because they are not specialized to meet the requirements of a specific type of visualization. However, this means that more work may need to be done to get them to produce simple standard visualizations that are already provided by the specialized tools.

The chosen high end tool was OpenDX. In addition to being a sufficiently powerful visualization tool, OpenDX incorporates a visual program editor (VPE) that allows users to visualize climate data and to directly manipulate the graphical output. The VPE includes a graphical user interface builder (GUI) so that OpenDX can also be used by software developers to build visualization applications for their users.
For the low end it was decided to use VCS (part of CDAT) as its Python interface and its alignment with the CF format means that it combines well with COCO, the chosen processing library, when writing processing and visualization scripts.

The VTK tool was also further developed as it held a potential both for being a high quality visualization tool suitable for use as a high end package, and for being incorporated as a module within CDAT that would enable it to be used in the low end graphics.

1.3 Processing with COCO

COCO (CDMS overloaded for CF Objects) is a data processing library. It is written in Python and is based on the Climate Data Management System (CDMS). CDMS is an object-oriented data managed system, specialized for organizing multidimensional, gridded data used in climate analysis and simulation.

The basic unit of computation in CDMS is the variable. A variable is essentially a multidimensional data array, along with its metadata describing the domain, grids and other attributes. As a data array, a variable can be sliced to obtain a portion of the data, and can be used in arithmetic computations. (For further in-depth discussion of CDMS see the CDAT homepage: http://esg.llnl.gov/cdat/)

See the Developer Documentation (Section 1.3.4) for instructions on how to install COCO. To use COCO, first start up the python interpreter and then import the ‘cdms’ module. This then provides access to all the cdms methods and those in the COCO layer.

$ python
>>> import cdms
>>>

In the COCO library, data and associated metadata belong together. They are read in from disk, manipulated and written out in conjunction. All input data files processed with COCO should adhere to the CF convention. If the files do not fully adhere to the convention then errors may occur, and the associated metadata may not be updated correctly. Any data written out to files will be CF compliant.

To check that any netCDF files are CF compliant, use the CF checker. This is available for download at http://prism.enes.org/WPs/WP4a/ProcessingLib/index.html or use it directly via the website http://titania.badc.rl.ac.uk/cgi-bin/cf-checker.pl

COCO enhances the CDMS functionality to include manipulation of the metadata. Additionally, COCO introduces the following functionality:

- **collapse(method=method, axis=axis, weights=weights, tolerance=tolerance)** Applies the chosen statistical methods to any combination of axes, with or without weights, and with or without a user specified tolerance to missing data. The methods implemented are: mean, mid-range, median, max, min, sum, standard deviation, variance, covariance, correlation.

- **textract(args, **kwargs)** Enables a time slice of data to be extracted from a single variable. For example, to extract all January’s data for all years.

- **getTimeComponents()** Returns a dictionary of date-time structures. For example, all the years, months, days, hours, minutes and seconds.

Full documentation for these three functions follows.
1.3.1 Detailed documentation of COCO methods

collapse()

Collapse is the function that interfaces with the statistical methods.

Usage

\[
\text{result} = \text{var.collapse(method=methodopts, axis=axisopts, weights=weightopts, tolerance=toleranceopts)}
\]

Options

methodopts:
- 'mean'
- 'median'
- 'max'
- 'min'
- 'mid_range'
- 'sum'
- 'sd'
- 'variance'
- 'covar'
- 'corr'

axisopts: 'x' | 'y' | 'z' | 't' | (dimension_name) | 0 | 1 | ... | n

The name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

weightopts:
- default = None returns equally weighted statistic.
- If you want to compute the weighted calculation, provide an array of weights (of the same shape as the dimensions being collapsed over or the same shape as var) here. Alternatively, specify a weighting type appropriate to the dimensions being collapsed over:
  - 't'
  - 'z'
  - 'y'
  - 'x'
  - 'northward_distance'
  - 'eastward_distance'
  - 'area'
  - 'simple'.

If the collapse is over more than one axis then a weighting option must be supplied for each dimension.

toleranceopts:
- default value = 1 No tolerance of missing data.
- Set to a fraction between 0.0 and 1.0 to specify the minimum weighted fraction of a collapsed cell which must have valid data in order for a representative value to be calculated.

Examples:

$ python
Python 2.2.2 (#1, Feb 26 2004, 11:13:44)
[GCC 3.2.2 20030222 (Red Hat Linux 3.2.2-5)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import cdms
>>> # Read in the variable 'tap' from a file
>>> tap=cdms.open('model.nc').getVariable('tap')
>>> # Calculate an area average lat/lon mean
>>> mean=tap.collapse(method='mean',axis='yx',weights='area')
>>> mean[0].shape
(22, 9, 1, 1)
>>> print mean[0].getValue()[0,0]
[ [219.071286768 ,]]
>>> # Open output file and write out the result
>>> out=cdms.open('/home/hadro/temp/out.nc','w+')
>>> out.write(mean[0])
<Variable: tap, file: /home/hadro/temp/out.nc, shape: (22, 9, 1, 1)>
1.3. PROCESSING WITH COCO

>>> # Read in one variable from a file that contains many and perform
>>> # standard deviation over the time axis
>>> tas=cdms.open('abefda.psj1djf.nc').getVariable('tas')
>>> tas
<Variable: tas, file: abefda.psj1djf.nc, shape: (1, 73, 96)>
>>> std=Collapse([tas], method='sd', axis='t')
>>> std[0].shape
(1, 73, 96)

If you wish to perform the same calculation on several variables of the same dimensionality, you can do
so with just one call to collapse. Instead of accessing collapse as a method of the variable, import the
dataoperations module and pass in the list of variables over which to calculate the statistic.

>>> import cdms, cdms.dataoperations
>>> # Read in several variables and perform a mean over y axis with
>>> # latitude weighting.
>>> # Note: to process several variables at once, they must all
>>> # have the same dimensionality.
>>> vars=cdms.open('abefda.psj1djf.nc').getVariables()
>>> vars
[<Variable: tasmax, file: abefda.psj1djf.nc, shape: (1, 73, 96)>,
 <Variable: tasmin, file: abefda.psj1djf.nc, shape: (1, 73, 96)>,
 <Variable: tas, file: abefda.psj1djf.nc, shape: (1, 73, 96)>]

>>> mn=cdms.dataoperations.Collapse(vars, method='mean',
                                  axis='(latitude)', weights='latitude')
>>> len(mn)
3
>>> mn[0].shape
(1, 1, 96)
>>> mn[0].getValue()[0,0,0]
281.58047358194989
>>> mn[1].getValue()[0,0,0]
276.36157311333551
>>> mn[2].getValue()[0,0,0]
278.86281840006512

More about specifying weights:

>>> tap.collapse(method='mean', axis='yx', weights='x')
is invalid since it doesn’t say how to weight y.

>>> tap.collapse(method='mean', axis='yx', weights='area')
is valid as weights are xy area weighted.

Currently, only a limited selection of weights have been implemented. They are:

- axis='x', weights='x' - weighting by difference betw. the bounds
- axis='y', weights='y'
- axis='z', weights='z'
- axis='t', weights='t'
- axis='yx', weights='area' - lat-lon area weighting
• axis='yx', weights='simple'
• axis='y', weights='northward_distance' - weighted by cell-widths in m

Other Useful Notes:
• See further documentation of the collapse method in the file cdms/dataoperations.py. The documentation can be viewed by running:
  
  pydoc dataoperations.py

• Whether collapse is called using (tap.collapse(...) or cdms.dataoperations.collapse([tap],...)) the result will always be a list of cdms variables. Even if the result is just one variable it will be as a list containing one variable. See first example above.

\textbf{textract()}

Reads in a slice of data, returning a transient variable.

\textbf{Usage:}

\begin{verbatim}
result=var.textract(*args, year=None, month=None, day=None)
\end{verbatim}

\textbf{Options:}

Optional Input:
‘args’ is an argument list of conditional strings. If this string is supplied, \texttt{textract} will append to it, starting with ‘and’. Otherwise, the resulting search expr is purely the result of the keywords. If more than one string is supplied they will be concatenated together with ‘and’.

Input Keywords:
The optional keyword arguments ‘year’, ‘month’, ‘day’ may be used to specify specific search criteria. For example to extract specific years, etc. If any of the above keywords is a list, this is by default converted into a list of OR’ed conditions.

\textbf{Examples:}

\begin{verbatim}
sp.textract (year=1990, month=[11,12])
\end{verbatim}

extracts data satisfying the search expression:
\begin{verbatim}
year = 1990 and (month = 11 or month = 12)
\end{verbatim}

\begin{verbatim}
sp.textract ('year < 1990', month = 12)
\end{verbatim}

extracts data satisfying the search expression:
\begin{verbatim}
year < 1990 and month = 12
\end{verbatim}

\begin{verbatim}
sp.textract (year=[1980,1990], month=[1,2,3])
\end{verbatim}

extracts data satisfying the search expression:
\begin{verbatim}
(year = 1980 or year = 1990) and (month = 1 or month = 2 or month = 3)
\end{verbatim}

\begin{verbatim}
sp.textract ('year >= 1980', 'year < 1990')
\end{verbatim}

extracts data satisfying the search expression:
\begin{verbatim}
year >= 1980 and year < 1990
\end{verbatim}
**getTimeComponents()**

Returns a dictionary of the time components, as lists of years, months, days, hours, minutes and seconds.

**Usage:**

\[
\text{tc} = \text{var.getTimeComponents()}
\]

**Examples:**

```python
>>> tc=tap.getTimeComponents()
>>> tc
{'second': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 'hour': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'year': [1974, 1975, 1976, 1977, 1978, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995], 'day': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], 'minute': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'month': [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9]}
```

**1.3.2 Tutorials and Examples**

A variable can be obtained from a file or collection of files, or can be generated as a result of a computation.

To open a file:

```bash
>>> f=cdms.open('file.nc')
```

To read in variable ‘tap’:

```bash
>>> tap=f.getVariable('tap') # or use tap=f['tap']
```

or alternatively in one step:

```bash
>>> tap=cdms.open('file.nc').getVariable('tap')
```

To read all variables in the file into one list:

```bash
>>> vars=cdms.open('file.nc').getVariables()
```

To write a variable to a file, use the write function:

```bash
>>> o = cdms.open('out.nc','w')
>>> o.write(tap)
```

To determine the shape of the data array:
Applying a mask from one file to a variable in another file and calculating an annual mean.

```python
>>> tap.shape
(22, 9, 73, 96)
```

```python
import cdms, MA
# Open file read
f=cdms.open('ustrw_e63_all.nc')
variab=f('ustrw')
data=MA.array(variab.getValue())
# Open and read in variable containing mask
maskVar=cdms.open('slm_e63_all.nc','r').getVariable('slm')
mask=MA.array(maskVar.getValue())
# Create new masked data array
new=MA.masked_array(data,mask=mask)
# Put the new data back into the variable
variab.putValue(new)
print variab.getValue()[1,0,0]

# Annual mean
import cdms
annual_mn=variab.collapse(method='mean',axis='t')
```

1.3.3 Links to reference materials

NetCDF(CF) website

http://www.cgd.ucar.edu/cms/eaton/cf-metadata/

CDAT homepage

http://esg.llnl.gov/cdat/

Documentation:

http://esg.llnl.gov/cdat/documentation.html

1.3.4 Developer Documentation

Download Information

All source code for the processing library developed under the PRISM project is available from the PRISM cvs server bedano.cscs.ch.

Once logged onto the PRISM cvs server using ssh, you can download COCO by running the following commands:

```
$ export CVS_RSH = /path/to/ssh
$ export CVSROOT = :ext:cvs@bedano.cscs.ch:/users/cvs
$ cvs checkout PRISM_Data
```

This will download the UDUNITS package and all the COCO code. Please follow the steps below to install.

Installation, portability and local configuration

In order to use the CF Data Processing Library the following are prerequisites:

1. CDAT-4.0
2. Either the python distribution packaged with CDAT or Python2.3 or later if CDAT installed using independent distributing of python.

3. UDUNITS with the python interface The Unidata units library, udunits, supports conversion of unit specifications between formatted and binary forms, arithmetic manipulation of unit specifications, and conversion of values between compatible scales of measurement. This package, complete with the python interface, is available from the PRISM cvs server.

Installing the CF Data Processing Library:
Once the CF Data Processing code has been downloaded; carry out the following steps:

1. Go to directory cf.data_processing
2. Replace the cdms python (.py) files (site-packages/cdms/*.py) with those in directory cf.data_processing/cdms
3. Copy the CF Standard Name table cf.data_processing/cdms/standard_name.xml to site-packages/cdms directory.
4. Copy directory oo and file oo.pth to site-packages.
5. Replace the genutil python (.py) files (site-packages/genutil/*.py) with those in directory cf.data_processing/genutil.

Unit Tests
Once the package has been installed it is recommended that the unit tests are run. The unit tests are in the directory cf.data_processing/tests/functional.
To run these tests issue the following command from within the directory:

$ python dataopsunittests.py

A typical output will be of the form:

....................
..checkCollapseTimeLevel
...checkCollapseLevel
........

Ran 40 tests in 68.366s

If there are any errors they will be indicated thus:

$ python dataopsunittests.py

.....F.......................checkCollapseTimeLevel
...checkCollapseLevel
.checkCollapseLongitude
......checkTExtract

======================================================================
FAIL: checkCollapseTime (__main__.AveragerTestCase)

Traceback (most recent call last):
 File "dataopsunittests.py", line 184, in checkCollapseTime
     assert axis.units == 'days since 1975'
AssertionError

Ran 40 tests in 63.769s
The output informs the user that the `checkCollapseTime` test has failed at line 184 and that the value of `axis.units` is not, for some reason 'days since 1975'.

The tests are split up into separate test suites, one suite for each functionality. For example, one suite tests `Averager`, another suite tests covariance functions.

It is easy to configure the test suite to run selected test suites. At the end of the `dataopsunittests.py` file is code of the format

```python
TestSuite1 = unittest.makeSuite(AveragerTestCase,'check')
TestSuite2 = unittest.makeSuite(VarianceTestCase,'check')
TestSuite3 = unittest.makeSuite(StdDeviatorTestCase,'check')
TestSuite4 = unittest.makeSuite(MinimizerTestCase,'check')

alltests = unittest.TestSuite((TestSuite1,
        TestSuite2,
        TestSuite3,
        TestSuite4,
    ))
```

The test suites included in the final `alltests` list are those that will be run. In the example above only `TestSuite1` (Averager) and `TestSuite2` (Variance) will be run. In short, tests can be added, removed or commented out as required.

### 1.3.5 Design notes and diagrams

Figure 1.1 shows the architecture of COCO.

![COCO Architecture](http://prism.enes.org/WPs/WP4a/ProcessingLib/requirements.html)

Outline specification for a CF Processing Library:

http://prism.enes.org/WPs/WP4a/ProcessingLib/requirements.html
1.3.6 Notes for developers

Known Problems

There is a problem with plotting variables using vcdat if the netCDF files contain auxiliary coordinates. It results in a runtime error stating that the “Grid lat/lon domains do not match the variable domain”. A workaround has been implemented that means that auxiliary coordinates are currently ignored.

Where to look to extend functionality

All of the user interface methods for the statistical functionality are contained in a python file called cdms/dataoperations.py. Addition of further statistical functionality including weights calculations should be made to this file.

Recommendations

At the current time there are a lot of additions to the standard CDAT release. It is intended that these changes will eventually be incorporated into the mainstream development of CDAT. Until then it is recommended to stay with the CDAT version 4.0b3. The intricacies and interaction between CDAT and the COCO layer are complex resulting in changes to CDAT having knock-on effect with COCO.

1.4 Accessing Remote Climate Data

The OPenDAP/DODS software framework for storing and accessing climate data has been explored. See the following website for details of the software and examples of how it has been used:

http://www.opendap.org/

Here, it describes OPenDAP as providing software which makes local data accessible to remote locations regardless of local storage format. OPeNDAP also provides tools for transforming existing applications into OPeNDAP clients (i.e., enabling them to remotely access OPeNDAP served data). OPeNDAP software is freely available.

With OPenDAP, a data archive is set up in a way that allows access to data within it using a URL in much the same way as one might access a web page. By qualifying the URL that points to a particular dataset, different parts of the dataset can be viewed.

The OPenDAP model of data storage encourages a highly distributed system where data is stored at the institution where it was created and distributed, using an OPenDAP server, from that institution. Scientists at other locations can download the data using URLs or using OPenDAP compliant tools and analyse the data with their preferred local application.

Examples of the usage of OPenDAP by the VTK Mapper application can be found in Section 1.8.3.

1.5 Data Visualization with OpenDX

1.5.1 Introduction

OpenDX is an open source high-end visualization tool that can be used to visualize simple data sets as well as to analyze complex, time-dependent data from disparate sources. It enables users to examine their data interactively via a graphical user interface. In addition, it provides the means to build interactive visualization applications. OpenDX provides functional modules to visualize data in many ways, to transform the data, and to interact with users and other applications. An application developed in OpenDX consists of a
network of interconnected modules. Parts of such networks can be stored as macros for reuse. Completely new modules can be added as well by means of the module builder.

The standard PRISM data format, NetCDF CF-1.0, is not supported by the standard distribution of OpenDX. Therefore, a set of modules and macros has been developed that enables OpenDX to read files in this format. These modules and macros are described in section 1.5.2 and their use is illustrated in section 1.5.3. This is followed with references to further information about OpenDX in section 1.5.4. The installation procedure for OpenDX and the additional modules is described in 1.5.5. The last two sections 1.5.6 and 1.5.7 describe the limitations of the current implementation of the modules and possible future enhancements respectively.

### 1.5.2 NetCDF CF-1.0 Import Modules

Table 1.1 lists all PRISM specific OpenDX modules. These modules appear in the “Import and Export” category in the visual program editor (VPE) of OpenDX. You should read the section on the OpenDX data model in the OpenDX users guide before using the NcCFReadComponent and NcCFCombine modules. The modules are described in the following sections. Each section starts with a description of the module’s functionality. This is followed by two tables describing the inputs and outputs of the module.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NcCFCombine</td>
<td>Combines separate components output by the NcCFReadComponent module into a OpenDX field or series object</td>
</tr>
<tr>
<td>NcCFFileInfo</td>
<td>Retrieves the value of several standard file global attributes from a NetCDF CF-1.0 file</td>
</tr>
<tr>
<td>NcCFInvertInZ</td>
<td>Swaps the (scalar) data and the Z coordinates of a field with three dimensional positions</td>
</tr>
<tr>
<td>NcCFLabels</td>
<td>Uses the value of several standard variable attributes to create a set of text strings that can be used as captions and coordinate labels in plots</td>
</tr>
<tr>
<td>NcCFListVariables</td>
<td>Lists variables contained in a NetCDF CF-1.0 file</td>
</tr>
<tr>
<td>NcCFReadComponent</td>
<td>Uses the data from a variable in a NetCDF CF-1.0 file to create OpenDX data, position, connection, and invalid positions component objects</td>
</tr>
<tr>
<td>NcCFRead</td>
<td>Is a macro that combines the NcCFReadComponent and NcCFCombine modules to read common types of variables</td>
</tr>
<tr>
<td>NcCFReplaceZ</td>
<td>Replaces the Z coordinates in the position component of a field with the data component from another field</td>
</tr>
<tr>
<td>NcCFSigmaCoordinate</td>
<td>Computes mid-layer depths from layer thickness data. This module is useful when only layer thickness data is available from an isopycnic coordinate ocean model</td>
</tr>
<tr>
<td>NcCFVariableInfo</td>
<td>Provides meta information about a variable</td>
</tr>
</tbody>
</table>

Table 1.1: PRISM specific OpenDX modules appearing in the Import and Export category.

**NcCFCombine**

This module combines its inputs to a field or series of fields. All inputs should be arrays or series of arrays. In the latter case, the series must all have the same length. The arrays of the data, position and invalid inputs must all have the same number of elements. The array provided to the connection input should only refer to positions that are contained in the positions array.
### 1.5. DATA VISUALIZATION WITH OPENDX

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>values</td>
<td>array or series</td>
<td>The array(s) that will be the data component of the field(s)</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>positions</td>
<td>array or series</td>
<td>The array(s) containing the position data. The number of positions should be equal to the number of data values</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>invalid</td>
<td>array or series</td>
<td>The array(s) indicating invalid positions in the data. The number of invalid positions should be equal to the number of data values</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>connections</td>
<td>array or series</td>
<td>A list of connections between data positions. All references to positions should be valid. In addition, the element type of the connections should match the length of the position vectors. These requirements are not checked by the module</td>
<td>No</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>field or series</td>
<td>A field or series of fields constructed from the components that were provided to the module</td>
</tr>
</tbody>
</table>

#### NcCFFileInfo

This module retrieves the value of the (optional) file global attributes as defined by the NetCDF CF-1.0 conventions. When an attribute is not defined in the file the corresponding output is set to the value “Unknown”. The module fails when the file does not exist or is not a NetCDF file.

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>string</td>
<td>The path to the file from which the information is to be retrieved</td>
<td>Yes</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conventions</td>
<td>string</td>
<td>The value of the global attribute “Conventions” or “Unknown”</td>
</tr>
<tr>
<td>title</td>
<td>string</td>
<td>The value of the global attribute “title” or “Unknown”</td>
</tr>
<tr>
<td>institution</td>
<td>string</td>
<td>The value of the global attribute “institution” or “Unknown”</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>The value of the global attribute “source” or “Unknown”</td>
</tr>
<tr>
<td>history</td>
<td>string</td>
<td>The value of the global attribute “history” or “Unknown”</td>
</tr>
<tr>
<td>references</td>
<td>string</td>
<td>The value of the global attribute “references” or “Unknown”</td>
</tr>
<tr>
<td>comments</td>
<td>string</td>
<td>The value of the global attribute “comments” or “Unknown”</td>
</tr>
</tbody>
</table>

#### NcCFInvertInZ

This macro converts a field of the form \( f(x, y, z) \) to a field of the form \( z(x, y, f) \). This is achieved without doing the actual inversion by exchanging the content of the scalar data component with the values of the third component of the position vectors. Only three dimensional scalar fields are modified. Other kinds of fields are passed without modification.

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>field</td>
<td>The field to be inverted</td>
<td>Yes</td>
<td>-</td>
</tr>
</tbody>
</table>
### NcCFLabels

This module creates labels for the data and coordinate system of a variable in a NetCDF CF-1.0 file. The labels consist of the value of the `long_name` or `standard_name` attribute or the variable’s name (in this order) followed by the unit in which the data of coordinate is expressed enclosed by square brackets. The outputs of this module can be used as a caption in a plot or as labels of coordinate axes.

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>string</td>
<td>The path to a NetCDF file</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>variablename</td>
<td>string</td>
<td>The name of the variable in the file</td>
<td>Yes</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>string</td>
<td>A description the variable</td>
</tr>
<tr>
<td>axislabels</td>
<td>string list</td>
<td>The labels for the axes of the coordinate system</td>
</tr>
</tbody>
</table>

### NcCFListVariables

This module creates a list of variables contained in a NetCDF CF-1.0 file. The variables included in the list can be constrained by kind and coordinate system. The kinds of variables correspond to those described in the NetCDF CF-1.0 conventions. This module is typically used in interactive programs, that are capable of visualizing only specific kinds of data, to present a list of variables from which the user can select one.

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>string</td>
<td>The path to the file to be examined</td>
</tr>
<tr>
<td>kind</td>
<td>string</td>
<td>The kind of variables to be included in the search.</td>
</tr>
<tr>
<td>coordinates</td>
<td>string</td>
<td>The kind of (spatial) coordinate system that the variables should have.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>string  list</td>
<td>The names of all variables satisfying the selection criteria</td>
</tr>
</tbody>
</table>
NcCFReadComponent

This module reads the data for one of the components of a field or series of fields. The start, end and increment inputs are only meaningful when the variable is time dependent. Otherwise, these inputs are ignored. Note that time steps are counted starting from one.

According to the CF-1.0 conventions a variable’s coordinate system can be described using coordinate variables as well as auxiliary coordinate variables. The preference input determines which coordinate variables are used to compute the positions component of a field when both types of coordinates are available.

The outputs of several instances of this module can be combined into a field (or series of fields) using the NcCFCombine module. The NcCFRead macro is such a combination that creates a complete field from a NetCDF variable and its associated information in one step.

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>string</td>
<td>The path to a NetCDF file</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>variablename</td>
<td>string</td>
<td>The name of a variable in the file</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>what</td>
<td>string</td>
<td>The component for which the data should be retrieved. Valid values are “Values”, “Invalid positions”, “Positions”, “Connections”</td>
<td>No</td>
<td>“Values”</td>
</tr>
<tr>
<td>start</td>
<td>integer</td>
<td>The first time step for which data should be read</td>
<td>No</td>
<td>First time step</td>
</tr>
<tr>
<td>end</td>
<td>integer</td>
<td>The last time step for which data should be read</td>
<td>No</td>
<td>Last time step</td>
</tr>
<tr>
<td>increment</td>
<td>integer</td>
<td>The time step interval at which data should be read</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>preference</td>
<td>string</td>
<td>The preferred coordinate system type. Valid values are “Auxiliary” and “Coordinates”</td>
<td>No</td>
<td>“Auxiliary”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>array or series</td>
<td>The data read. It is an array when the variable is time independent. Otherwise, this output is a series containing array objects</td>
</tr>
</tbody>
</table>

NcCFRead

This macro combines several instances of the NcCFReadComponent module with one instance of the NcCFCombine module to create a field or series of fields from a variable in a NetCDF CF-1.0 file. It handles most cases except for variables with irregular connections. In the case of irregular connections, the read_connections input should be set to 0. OpenDX’s Connect module can be used to compute connections using a triangulation algorithm.
### NcCFReplaceZ

This macro replaces the Z coordinate of the position vectors of one field by the (scalar) data component of another field. It can be used to create fields that describe the relation between two quantities as a function of longitude and latitude without interpolation. Note that both input fields must have the same amount of data points.

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>filenames</td>
<td>string</td>
<td>The path to a NetCDF file</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>variablename</td>
<td>string</td>
<td>The name of a variable in the file</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>start</td>
<td>integer</td>
<td>The time step number of the first frame to read</td>
<td>No</td>
<td>First time step</td>
</tr>
<tr>
<td>end</td>
<td>integer</td>
<td>The time step number of the last frame to read</td>
<td>No</td>
<td>Last time step</td>
</tr>
<tr>
<td>increment</td>
<td>integer</td>
<td>The interval used to select frames from a time dependent variable</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>read_connections</td>
<td>integer</td>
<td>When this input is set to 1 the macro adds the connections components to the output but when this input is 0, the connections are not included</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>preference</td>
<td>string</td>
<td>The preferred kind of coordinate system. Valid values are “Auxiliary” and “Coordinates”</td>
<td>No</td>
<td>“Auxiliary”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>field or series</td>
<td>An OpenDX object constructed from the specified variable</td>
</tr>
</tbody>
</table>

### NcCFSigmaCoordinate

This module computes the mid-layer depth from layer thickness data produced by isopycnic ocean models as follows: let $dp_i$ be the thickness and $p_i$ the mid-layer depth of layer $i$, then $p_k = \sum_{i=1}^{k} dp_i - dp_k/2$. The output of this module can be passed to the NcCFReplaceZ macro to replace the z coordinate in positions data read using the NcCFReadComponent module. In this manner one can easily convert a field with isopycnal data to one with spatial data without interpolation.

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>thicknesses</td>
<td>field or series</td>
<td>The layer thicknesses. This input must be a (series of) regularly connected field(s)</td>
<td>Yes</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>depths</td>
<td>field or series</td>
<td>The mid-layer depths of the sigma layers at each position</td>
</tr>
</tbody>
</table>
1.5. DATA VISUALIZATION WITH OPENDX

The simplest OpenDX network with which some data can be visualized consists of three modules connected together (see figure 1.2). The NcCFRead module converts a variable in a CF-1.0 compliant NetCDF file to an OpenDX object. The name of the file and variable is specified in the configuration panel of the module which can be opened by double clicking on it. The OpenDX object needs to be colored before it...
can be rendered. This is done using the AutoColor module which applies a default colormap to the data. In this example a RubberSheet module is used just before the Image module to given the two-dimensional data a three-dimensional appearance. The output of the network is shown in the lower panel of figure 1.2. The Options menu enables the user to manipulate the rendered object (rotate, zoom, etc.), add coordinate axes and modify the background color. The image can be printed and saved with options in the File menu. As a next step we want to use a different colormap. The modified network is shown in figure 1.3. The data to be visualized is a bathymetry dataset with values giving the depth of the ocean at each grid point. We convert these to a height coordinate using the Compute module which applies a mathematical expression to each data value. The result of the Compute module is directed to a Colormap module and a Color module. The latter applies the colormap computed by the former to the data. The OpenDX colormap editor can be opened by double clicking on the Colormap module. Colormap data can be read and saved using options in the File menu of the colormap editor. The colormap can be modified with several menu items under Edit and by means of the control points in the Hue, Saturation, Value and Opacity columns. Minimum and maximum values can be specified in the editor or can be computed from the data of an OpenDX object that is input to the Colormap module. The output of the modified network is shown in last panel of figure 1.3.

As a final improvement we add coordinate axes to the image. For this we need to set the scale factor of the RubberSheet module to one using its configuration dialog. As a result this module computes z coordinates in meters. The object created by the RubberSheet module now has an x:y:z aspect ratio of 360:180:5500 which makes the vertical size excessively large. We use a Scale module to scale the z coordinate to obtain a more pleasing aspect ratio (see figure 1.4). The resulting image is shown in the lower right panel. The coordinate axes were added by opening the AutoAxes dialog from the Options menu and entering the required information as shown in the figure. Note that it is also possible to let the additional modules compute this information and supply it as input to the Image module. The inputs to the Image can be made visible in the network by opening the configuration panel of the Image module (select the module and press Ctrl-F) and clicking the Expand button at the bottom of the panel. The check boxes in the Hide column can be unchecked to let the corresponding input tab appear in the module’s box in the network.

The second example illustrates some of the advanced visualization capabilities of OpenDX. The OpenDX network of this example is to large to be shown here. It is included with the source code distribution the NetCDF CF-1.0 import modules. The data set that is being visualized consists of two files: one with volumetric data and one of surface data produced by the ECHAM atmosphere model. Figure 1.5 shows what the main visualization looks like. The upper half shows the volumetric data on a longitude, latitude and pressure grid. Cloud cover is shown as spheres with sizes corresponding to the cloud cover fraction at each grid point. Wind vectors and air temperature are shown on a horizontal and a vertical slice. These slices can be moved to any vertical and longitudinal position respectively. The data on the slices are also shown as two dimensional plots in a separate window (not shown). The wind field is visualized using streamlines as well and the streamlines are colored according to the vertical velocity. Red and yellow indicate sinking air and blue and cyan rising air. The lower half of the display shows surface data as a stack of three levels. The topography is shown at the bottom with colors indicating the surface pressure. Above it the evaporation and precipitation are shown. In both cases the height of the surface and the color indicate the amount of evaporation and precipitation. The selection of data that is visualized can be changed in a control panel that is not shown here. The OpenDX Sequencer module is used to create an animation of how the atmosphere changes in time.

1.5.4 References

The OpenDX website, http://www.opendx.org, provides on-line access to all user and programmer documentation and mailing lists, and has links to additional macro packages. The OpenDX quick start guide describes how you can start using OpenDX. Further information about using OpenDX and
Figure 1.2: A simple OpenDX network to visualize two dimensional data. The upper panel shows the network itself as it was entered in the editor, the center panel shows the configuration dialog for the NcCFRead module, and the output of the network is shown in the lower panel.
Figure 1.3: The modified network that applies a custom, editable colormap to the data. The upper left panel shows the network, the upper right panel the configuration dialog for the Compute module, the lower left panel the colormap editor, and lower right panel the output of the network.
Figure 1.4: A network to visualize data with coordinate axes. The upper left panel shows the network, the upper right panel the configuration dialog for the Scale module, the lower left panel the output, and lower right panel the dialog used to configure the axes.
Figure 1.5: Output of the example network showing the advanced visualization capabilities of OpenDX.
its programming language can be found in the Users Guide. The standard modules are described in the Reference Guide. The highlights section of the OpenDX site might give you some ideas for visualizing your data. Its bookstore provides a list of books about OpenDX and visualization.

A macro package that is of interest to climate models is the Earth and Space Sciences package available at Cornell Theory Center (CTC, http://www.tc.cornell.edu). The CTC also has an on-line introduction to OpenDX.

The Department of Oceanography at Dalhousie University provides a tutorial showing about advanced visualization with OpenDX (http://www.phys.ocean.dal.ca/docs/DXtutorial.html). VISUALIZATION and IMAGERY SOLUTIONS provides commercial support.

1.5.5 Installing OpenDX and NetCDF CF-1.0 Modules

In this section we describe how OpenDX and the NcCFModules can be installed. We assume that you have downloaded the source code packages of both of these from the PRISM site, that you are using a Linux PC, and that the NetCDF and udunits libraries have been installed before in the (non-standard) directories /usr/local/netcdf and /usr/local/udunits. In this case you can configure, compile and install OpenDX in your home directory by entering the following commands (assuming that you are using bash)

```bash
export CPPFLAGS="-I/usr/local/netcdf/include"
export LDFLAGS="-L/usr/local/netcdf/lib"
./configure --prefix=$HOME/dx
make
make install
```

Note that when you are using tcsh instead of bash you should replace export by setenv and the equal sign by a space in this and the following command examples. If the NetCDF and udunits libraries are installed in a standard location the export command lines can be skipped altogether. Before you can test the installation of OpenDX by running it you need to include the directory $HOME/dx/bin in the search path of the shell

```bash
export PATH=$HOME/dx/bin:$PATH
```

To start the visual program editor of OpenDX enter the command

```
dx -edit
```

Once OpenDX is installed, you are ready to configure, compile and install the NcCFModules. You can do this by entering the following commands

```bash
export CPPFLAGS="-I/usr/local/netcdf/include -I/usr/local/udunits/include"
export LDFLAGS="-L/usr/local/netcdf/lib -I/usr/local/udunits/lib"
./configure --with-dxroot=$HOME/dx
make
make install
```

As before you can skip the export command lines when the NetCDF and udunits libraries are installed in a standard location. Before you can use the new modules in OpenDX you should set some environment variables in your .bashrc file

```
dxroot=$HOME/dx
export DXMACROS=$dxroot/dx/macros
export DXMDF=$dxroot/dx/modules_linux/NcCFModules.mdf
export DXMODULES=$dxroot/dx/modules_linux
unset dxroot
```

or in case of tcsh you should add the following to your .tcshrc file

```
set dxroot=$HOME/dx
```
setenv DXMACROS .:$dxroot/dx/macros
setenv DXMDF $dxroot/dx/modules_linux/NcCFModules.mdf
setenv DXMODULES .:$dxroot/dx/modules_linux
unset dxroot

Note that on non-Linux systems the directory names containing \texttt{linux} should be replaced by the names corresponding to the operating system you are using. You can determine which names to use by listing the content of the $HOME/dx/dx directory when OpenDX is installed in $HOME/dx. You can test whether the modules have been installed successfully by starting OpenDX with

dx -edit

The NcCFModules should be listed in the “Import and Export” category if installation was successful.

1.5.6 Limitations

The visualization of climate model data using the advanced capabilities of OpenDX requires large amounts of computer resources. This implies that for optimal performance all data is best stored on local file systems and available computing power should be allocated to the rendering of the data as much as possible. The latter is the reason that COCO has not been integrated into OpenDX although this is technically feasible. Of course files produced by COCO can be imported into OpenDX because they conform to the NetCDF CF-1.0 conventions. Since the NetCDF CF-1.0 modules access data files only via the NetCDF library, it should be possible to access remote files by linking the modules to the DODS netCDF client library. However, this has not been tested yet.

The NetCDF CF-1.0 modules for OpenDX are able to read all NetCDF files that conform to the NetCDF CF-1.0 conventions. However, in the current implementation the cell boundary information is ignored. These cell boundaries provide information about the connectivity of the grid. In the case of regularly connected grids - a common situation in data produced by climate models - the cell boundary information is redundant. In the case of irregularly connected grids the \texttt{Connect} module of OpenDX can be used to compute a triangulation of grid. An advantage of using the information on cell boundaries is that it most likely provides a faster way of determining the grid connectivity.

1.5.7 Future Developments

We foresee two functional improvements of the current set of modules. First, the NcCFReadComponent can be extended to use the cell boundary information when available to compute the connectivity of a field. Second, a new module can be added that interfaces to the COCO data p-processing tool and enables one to execute COCO scripts from OpenDX. In addition, the modules can be tested with the DODS netCDF client library. In case the NetCDF CF conventions are updated the modules should modified to support these changes.

1.6 Low End Visualization

1.6.1 Introduction

To enable the user to quickly survey the output data during the model run, groups of diagnostic pictures are created on the server and mounted to an html page. This mode of operation is different to high end visualization because the latter typically needs the data and applications to be on the local system due to the need for interactivity (figure 1.6).

This chapter describes: the tools for the picture generation (CDAT and VCS); the script language (Python) to control the process of Low End Visualization; the Python scripts of the Low End Graphic Package; the script to mount these pictures on an html page; the ParaGen automatic code generator; a colour editor to generate colour maps.
Figure 1.6: Low End visualization is server side generated, whereas data for High End visualization is downloaded to the client.
Climate Data Analysis Tools (CDAT) is a software infrastructure that uses an object-oriented scripting language to link together separate software subsystems and packages thus forming an integrated environment for solving model diagnosis problems.

The power of the system comes from Python and its ability to seamlessly interconnect software. Python provides a general purpose and full-featured scripting language with a variety of user interfaces including command-line interaction, stand-alone scripts (applications) and graphical user interfaces (GUI).

The CDAT subsystems, implemented as modules, provide access to and management of gridded data (Climate Data Management System or CDMS); large-array numerical operations (Numerical Python); and visualization (Visualization and Control System or VCS).

One of the most difficult challenges facing climate researchers today is the cataloging and analysis of massive amounts of multi-dimensional global atmospheric and oceanic model data. To reduce the labor intensive and time-consuming process of data management, retrieval, and analysis, PCMDI and other DOE sites have come together to develop intelligent filing system and data management software for the linking of storage devices located throughout the United States and the international climate research community.

This effort, headed by PCMDI, NCAR, and ANL will allow users anywhere to remotely access this distributed multi-petabyte archive and perform analysis.

PCMDI’s CDAT is an innovative system that supports exploration and visualization of climate scientific datasets. As an “open system”, the software sub-systems (i.e., modules) are independent and freely available to the global climate community. CDAT is easily extended to include new modules and as a result of its flexibility, PCMDI has integrated other popular software components, such as: the popular Live Access Server (LAS) and the Distributed Oceanographic Data System (DODS). Together with ANL’s Globus middleware software, CDAT’s focus is to allow climate researchers the ability to access and analyze multi-dimensional distributed climate datasets.

Visualization and Control System (VCS) is expressly designed to meet the needs of climate scientists. Because of the breadth of its capabilities, VCS can be a useful tool for other scientific applications as well.

VCS allows wide-ranging changes to be made to the data display, provides for presentation hard copy output, and includes a means for recovery of a previous display.

In the VCS model, the data display is defined by a trio of named object sets, designated the “primary objects” (or “primary elements”). These include:

- the data, which define what is to be displayed and is ingested via other CDAT software components; the graphics method, which specifies the display technique; and the picture template, which determines the appearance of each segment of the display.

Tables for manipulating these primary objects are stored in VCS for later recall and possible use. In addition, detailed specification of the primary objects’ attributes is provided by eight “secondary objects” (or “secondary elements”):

1. colormap: specification of combinations of 256 available colors
2. fillarea: style, style index, and color index
3. format: specifications for converting numbers to display strings
4. line: line type, width and color index

5. list: a sequence of pairs of numerical and character values

6. marker: marker type, size, and color index

7. texttable: text font type, character spacing, expansion and color index

8. textorientation: character height, angle, path, and horizontal/vertical alignment

By combining primary and secondary objects in various ways (either at the command line or in a program), the VCS user can comprehensively diagnose and inter-compare climate model simulations. VCS provides capabilities to:

- Create and modify existing template attributes and graphics methods

- Save the state-of-the-system as a script to be run interactively or in a program

- Save a display as a Computer Graphics Metafile (CGM), GIF, Postscript, Sun Raster, or Encapsulated Postscript file

- Create and modify colormaps

- Zoom into a specified portion of a display

- Change the orientation (portrait vs. landscape) or size (partial vs. full-screen) of a display

- Animate a single data variable or more than one data variable simultaneously

- Display different map projections

1.6.3 Python

**Home page:** http://www.python.org

**Documentation:** http://www.python.org/topics/learn/

Python is an interpreted, interactive, object-oriented programming language. It is often compared to Tcl, Perl, Scheme or Java.

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, as well as to various windowing systems (X11, Motif, Tk, Mac, MFC). New built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface.

1.6.4 The ParaGen Code Generator

Scripts used to process data and generate plots for the low end system may be used in other contexts such as within a test harness that is potentially written in a range of different scripting languages, or from a UNIX command line. The requirements for how scripts are called differ in each of these situations. To
minimize redundancies and inconsistencies between different instances of a script an the automated code generator ParaGen was developed. ParaGen defines an XML schema for describing scripts which includes a description of the interface and documentation for the script. For the Low End Visualization the \texttt{LE\_parameter.py} Python script was written in an XML file conforming to this schema.

![Figure 1.7: Once the syntax of a command is defined in an xml file, ParaGen allows for automated code generation. Given such a file, ParaGen can generate several different types of output shown schematically in figure 1.7:](image-url)
1.6. LOW END VISUALIZATION

- an html help file describing the script. See, for example:
  
  http://prism.enes.org/
  WP4a/Low-End/Documentation/standalone/sta.home.html

- an application specific wrapper code that checks syntax of the input (currently operational for the languages c and Python),

- an html form so users can control the script by inputting parameters in a web page. Here, for example, if a valid input for an argument to the script is one of a list of values, they will be coded in the html as pull down menus,

- an application specific output containing help information for use in systems such as a command line manual page systems (currently operational for the languages c and Python).

- Additionally, an xml output file is generated, that guides the Java code generation for those parts of the prepIFS user interface, where the parameter input by the user is expected.

1.6.5 The XML colour editor

During the project the need for a common color table file format, which can be used with different visualization applications, was identified. Color tables are needed for simple 2d visualizations (eg. contour maps), as well as for complex 3d visualizations. It would be very helpful to be able to use the same colour scheme for both cases. Imagine comparing imagery created by visualization software A with images produced by visualization software B! Just a dream?

The search on the internet for already existing color table definitions for scientific applications did not reveal a general and publicly available format. Every visualization system has its own way to define the colour maps and the mapping between the data and the colours. So we started to develop an XML based color table schema ourselves, aiming for an easy to use but powerful definition. Index color tables should be supported as well as color tables for pseudo-color visualizations. We found that the XML file format is the obvious choice for a general approach.

The schema we have defined will hopefully be useful for a wide range of applications. As a visual introduction to our concept, we have written a little application (Java Web Start required) where xml color tables can be created, modified, displayed, saved and loaded. See figure 1.8.

The strength of any data format is not only in the definition itself, but in the popularity of the format. The more people and applications that are using the same format, the more useful it is. Therefore we would like to share these results of our work with the community, and we would like to see our format used by as many applications as possible!

See:

http://www.ppart.de/apps/color/

for the code and further information.

Output example of the XML colour editor:

```xml
<color-table mapping-exponent="constant" name="RGB"
    mapping-color-space="rgb">
  <color>rgb24(255,0,0)</color>
  <color>rgb24(0,255,0)</color>
  <color>rgb24(0,0,255)</color>
</color-table>
```

PICTURE: JCTColorChooser.eps CAPTION: With the Java based ColorEditor, the user can build specific colour tables and store them in a generic format (XML).
With the Java based ColorEditor, the user can build specific colour tables and store them in a generic format (XML).

1.6.6 Low End Graphic Package

Main program and modules

The Low End Graphics Package allows the user to create standard plots without the need of knowledge about the underlying graphics software CDAT/VCS.

The user must only enter the input file name and the variable to display the data from a netCDF-CF file created by the PRISM system when calling the ksh script LE\_parameter.py.

The data could be displayed with isolines, isolines filled (isofill), boxes filled (boxfill), vectors (vector) and lines (e.g. for time series). The standard plot modules define the default values for a number of colors to be used, the color table, the isoline/isofill interval and more.

Inside PrepIFS most of these settings can be changed by the user. Running the Low End Graphics Package stand alone on a local machine the users have more possibilities to change these settings using the different options of the program (see Low End Manual Page).

For an extended set of examples see the enes.org web page at http://prism.enes.org/WPs/WP4a/Low-End/Documentation/standalone/sta.leuse.html.

Examples of Program calls:

To display, e.g., the sea surface temperature SST of a MPI\_OM data file using the color table Blue-ToRed16:

LE\_parameter.py -i "SSTOCEAN\_out.0001-01-01T00:00:00.nc" -var "SSTOCEAN" -p "linear" -m -colors "BlueToRed16"

To display the 2m temperature with vector overlay of u-v wind:
LE_parameter.py -i "BOT_197801.nc" -var "temp2" -dint "2.5" -p "linear" -gol "vector" -gov "u10:v10" -colors "Rainbow32"

Syntax

    LE_parameter.py
    [ -h | -H ]
    [ -i |--input String
      -var |--variable String
    [-gxo |--gxout GraphicMethods ]
    [-p |--projection Projections ]
    [-lat |--latitude float float ]
    [-lon |--longitude float float ]
    [-gridfile |--grid-file String ]
    [-bounds |--bounds-latlon String String ]
    [-gol |--goverlaymethod GraphicMethods ]
    [-gov |--govervariable String ]
    [-nc |--numberofcontours int ]
    [-dint |--contourintervall int ]
    [-l |--level String ]
    [-t |--time String ]
    [-m |--mesh String ]
    [-mask |--mask String ]
    [-mvar |--mvar String ]
    [-u |--units String ]
    [-or |--orientation Orientations ]
    [-pr |--print_format Printformats ]
    [-user |--username String ]
    [-colors |--colortable String ]
    [-bg |--background String ]
    [-tser |--time-series ]
    [-notext |--without-header-text ]
    [-yrev |--yreversed ]
]

Usage:
-h, -H
Print out this help information. Using this option will end the program.

-i String, -input String
input data file name
(required)

-var String, -variable String
Name of the data variable to be displayed.
(required)

-gxo GraphicMethods, -gxout GraphicMethods
Name of the graphic method to be used
Allowed values for GraphicMethods = boxfill,isofill,isoline,vector

-p Projections, -projection Projections
The projection to be used.
Figure 1.9: An example of an irregular grid, plotted by LE-parameter.

Allowed values for Projections = linear,polar,robinson,mollweide

-lat float float, –latitude float float
Select latitude subregion.
Default: -90:90

-lon float float, –longitude float float
Select longitude subregion.
Default: -180:180

-gridfile String, –grid-file String
External grid file.

-bounds String String, –bounds-latlon String String
Name of the bounds lat,lon variable of the grid file.

-gol GraphicMethods, –goverlaymethod GraphicMethods
Graphics method of overlayed data variable. Allowed values for GraphicMethods = boxfill, isofill, isoline, vector

-gov String, –govervariable String
Name of the overlayed variable.

-nc int, –numberofcontours int
Number of contour line levels.
Default: 15

-dint int, –contourintervall int
Intervall of contour lines in hPa.
Default = 5

-l String, –level String
Set level to be used.
Default: 1

-t String, –time String
Time value to be used.
Default: 1

-m String, –mesh String
Make meshfill and grid plot.
-mask String, –mask String
Name of the netCDF-CF file including the mask data.

-mvar String, –mvar String
Name of the mask variable.

-u String, –units String
Units: change Kelvin data values to Celsius.

-or Orientations, –orientation Orientations
Change the orientation of the plot output.
Allowed values for Orientations = portrait, landscape

-pr Printformats, –print_format Printformats
Change the output print format.
Allowed values for Printformats = gif, ps

-user String, –username String
Print user name on the plot.
Default: unknown user

-colors String, –colortable String
The xml file of the colortable.
Default: VCS rainbow

-bg String, –background String
Background (foreground=0, background=1)
Default: 1

-tser, –time-series
Time Series

-notext, –without-header-text
Plot only the graphic without title and comments.

-yrev, –yreversed
Y reversed plot.

Error Codes:

0   OK
900 opening failed - file not found
901 closing failed
910 creating mesh failed
911 calculation failed
920 cannot initialize VCS graphic
921 cannot open VCS canvas
930 colortable file not found
931 unable to create the plot
932 writing print file failed
992 Out of Memory during argument parsing
993 Parameter is not a boolean:
994 Parameter is not a character:
995 Parameter is not a number:
996 Required Argument Missing:
997 Illegal Parameter at:
998 Missing parameter at:
999 Illegal parameter value for enumeration type:
1.6.7 Generation of Overviews (Thumbnails and html pages)

General Remark

This Python script has been designed to create small thumbnail pictures from a given set of full size pictures. The thumbnails are mounted on one or more html pages. Click on a thumbnail leads to the appropriate full size picture. An additional index page links to all picture pages.

The thumbnails are mounted on nPag html pages, each containing nTab tables with nRow * nCol cells (nRow and nCol are the numbers of table rows and columns, respectively). So the thumbnails span a space of up to four dimensions, they are distributed on the web pages according to the names of the full size pictures.

For instance, the pattern of the input files can read, e.g., AAAmyfileBccDD.jpg where ABcD are four proxies, that span four dimensions in the picture file names. The script will look for the full size picture files by the command ls ???myfile?????.jpg. They will be converted to thumbnails and mounted on .html pages.

Figure 1.10: An example of the thumbnail layout, generated from the LE parameter output by tt4www.

Syntax

```
tt4www.py
   -p  fileNamePattern
   -q  proxyCharacters
       [-cc  compressionCommand ]
       [-cp  compressionParameters ]
       [-hp  preHtml ]
       [-fp  prePic ]
       [-tp  preThumb ]
       [-fe  postPic ]
       [-te  postThumb ]
```

-q A string of up to four proxy characters.
A proxy can be any character that is not part of any of the file names (of the files to be visualized). The order of the characters in this string determines the numbers of columns, rows, tables, and pages.
-p The pattern of the picture file names.
All proxies must be part of this pattern, all other characters of this pattern will be regarded to be fixed, i.e.,
to occur in all file names at this position.

-cc The compression command to use. “none” for no compression.
Here a filter can be specified to generate small pictures. Default: convert. The convert command belongs
to the Imagemagick.org package.

-cf Compression forcing.
With this parameter, the compression step is called for each full size picture file.
Default: no compression if target file exists already.

-cp The parameters to use for the compression command.
Any parameters to be placed directly after the compression command. Under Unix you will probably have
to mask this: e.g., ‘-geometry "10%"’
Default: geometry "25%"

-hp A prefix/PATH for the html pages to be generated.
Here path and name of the *.html can be given.
Default: ./tt4www

-fp A PATH for the full size picture files.
Here the path of the full size picture files can be given.
Default: ./

-tp A prefix/PATH for the thumbnails.
Here a path and a name prefix of the thumbnails can be given.
Default: ./

-fe A postfix/extension of the full size picture files.
Here an extension for the input files can be given.
Default: .gif

-te A postfix/extension of the thumbnails.
Here an extension for the output files can be given.
Default: .png

Examples:

1. Short Example

python tt4www.py -p AAAmyfileBccDD -q AcDB
This command will convert all files ???myfile?????.gif to thumbnails and mount them on html
pages.
The number of html pages is equal to the number of different characters at the B-position of the pattern.
The number of rows of a table is equal to the number of different patterns at the DD-position of the pattern.
The number of columns of a table is equal to the number of different patterns at the AAA-position of the pattern.

2. Files of 3 years (2001..03), 12 months, 3 variables (A,B,C), and 2 picture modes (BaW, COL)

python tt4www.py -p YYYY-MM_v_mmm -q vmMY -fe .jpg

The filenames are according to the pattern YYYY-MM_v_mmm.jpg, e.g., file 2002-09_C_BaW.jpg contains the black and white picture of parameter C in September 2002. The tt4www.py script will produce 3 html pages, one for each year. On each page there will be 12 tables, one for each month. Each
table will have three columns, one for each parameter \(A, B, C\), and two rows, the upper one black and white, the other in colour.

### 1.7 VTK

The Visualization ToolKit (VTK) is a widely used C++ library in the graphics, visualization and imaging domain. VTK consists of an API (Application Programming Interface) with more than 700 C++ classes implemented with more than 350,000 lines of C++ code (110,000 executable lines) and with more than 215,000 lines of automatically generated Python wrapper code. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as implicit modelling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. In addition, dozens of imaging algorithms have been directly integrated to allow the user to mix 2D imaging, 3D graphics algorithms and data. The design and implementation of the library has been strongly influenced by object-oriented principles. VTK has been installed and tested on nearly every Unix-based platform, PCs (Windows 98/ME/NT/2000/XP), and Mac OSX.

### 1.8 The VTK\_Mapper Application

The application VTK\_Mapper described in the remainder of this section is a prototype of an application made to demonstrate the ability of VTK, when used with other complementary packages, to meet the requirements for PRISM visualization. The initial goal was to demonstrate the high potential in designing applications using the VTK library to handle the visualization, the CDMS/COCO libraries to access and process netCDF files and the Qt library to build handy and nice user interface.

Designing and implementing such a prototype is helpful in ensuring that essential features such as 2D and 3D rendering, vector-based quality output, batch and off screen running mode, are fully covered. Such an application is then suitable for both Low End environment use (scripting, batch and off screen modes) and High End use (high interactivity).

#### 1.8.1 Software architecture

All the code of the VTK\_Mapper application has been written in the Python language which is a portable open source scripting language. This powerful programming language can be extended with compiled modules implemented in C/C++ or FORTRAN. To extend and bring higher visualization capabilities to the CDAT/VCS system, the Visualization ToolKit (VTK) has been investigated.

To read the netCDF files and access to all the metadata informations, the CDMS Python module has been employed. Its capacities to get the grid and the mesh from a variable stored in a netCDF file have been greatly helpful. The management of masked values recovered from the different possible combinations of masks (ocean/land mask and the variable mask itself) has been made possible through the use of the CDMS/MV Python module.

To design a user interface, the Qt library and its full set of GUI (Graphical User Interface) controls has been preferred to other libraries also explored. Qt is a platform-independent set of C++ classes that can be freely used in open source projects. It also comes with development tools such as Qt Designer to visually build your application. Technically, the VTK\_Mapper application has been implemented with calls to the Python bindings of the Qt toolkit, module called PyQt.

#### 1.8.2 Covered features

With the software architecture used in the VTK\_Mapper application, you can do the following:
1.8. THE VTK Mapper Application

**Read netCDF files using Climate and Forecast (CF) or COARDS convention.** By using the CDMS module, the reading of variables from netCDF files and the recovery of metadata informations have been easily managed. The Climate and Forecast (CF) convention used to store grid information of rectilinear, curvilinear and generic models and the corresponding GetGrid() and GetMesh() methods from the CDMS module have been particularly useful. CDMS also offers backward compliance for netCDF files using the COARDS convention. Conversions from one grid type to another are easily implemented through calls to the toCurveGrid() and the toGenericGrid() methods.

To let a user explore repositories of model output with conventions older than the CF convention, an external gridfile is written containing all information of the grid which could be missing in models output files. The VTK Mapper is then a really flexible and backward compliant application.

**Read remote netCDF files served by an OPeNDAP/DODS server.** By using the CDMS module compiled with the OPeNDAP netCDF API rather than the usual netCDF API, it becomes possible to read and access part of a remote netCDF file. It opens a virtually unlimited world access to models or data output repositories served by OPeNDAP/DODS servers. An IPSL OPeNDAP/DODS server has been set during the PRISM project to serve and share IPCC simulations output computed from the IPSL coupled model. It has also been used to test netCDF files from OASIS coupler and particularly their full compliance to the netCDF CF convention.

**Read CDML files (collections of netCDF files).** By using the CDMS module and CDML (Climate Data Markup Language), you can aggregate split netCDF files and see them combined as a single dataset. CDML files are generated by the cdscan command provided with the CDAT distribution.

**Render 2D/3D objects.** By using the VTK API, it is possible to render 2D and 3D complex data structures. An orthographic projection has been computed to produce a 3D scene and a linear (plate caree) projection to produce a 2D scene. Interactive zoom and translation with easy mouse controls are the proposed features of the VTK Mapper application to get interactivity. The user can also interactively switch between projections passing from a 2D to a 3D rendering.

**Render large objects with appropriate level-of-details.** By using the VTK API and the use of level-of-details objects, the application can achieve acceptable rendering performance at the cost of lower-resolution representation. This is particularly useful during motions when the application renders large objects to maintain interactive frame rates.

**Produce isocontours or polygons maps.** By using the VTK API and the marching squares algorithm, it is possible to generate isocontours from generic, curvilinear and of course rectilinear grids. Switching from an isocontour rendering to a polygon map rendering is as easy as pressing a key. By this interaction, the user avoids the long cycle of changing parameters/launching application.

Additional controls to pass from a vertical level to another or from a time level to another are proposed with the VTK Mapper application. The rendered modes (isofilled, cells, cellsbounds, isolines1, isolines2) are updated following key events.

Isocontours are generated with respect to the original topology of cells boundaries; other isocontour algorithms using cell centers have been investigated in particular Delaunay triangulation.

**Handle any type of model grid.** By using the VTK API, you can handle structured (uniform rectilinear, non-uniform rectilinear, and curvilinear grids), unstructured, polygonal and image data. The different dataset structures proposed by the VTK API cover all the needs for building a visualization application taking care to represent correctly topology and connectivity. The different model grids have been represented with the use of the vtkPolydata dataset type. This unstructured dataset type requires an explicit description of cells and points from the model grid. The connectivity is then dynamically computed with the use of a vtkCleanPolyData filter to join cells with shared boundary points.

**Probe variable values.** By using the VTK API, the user can focus on a particular zone and also probe values from the field displayed. This feature is particularly helpful when model codes are in a beta stage and when the user expects to examine the model output at its real and computed form.
Write raster output. By using the VTK API, it is possible to create raster images from the displayed window in different formats: Windows Bitmap (*.bmp), JPEG Images (*.jpg), PNG Images (*.png), Binary PPM (*.ppm) and TIFF images (*.tif). In the VTK_Mapper application the PNG image format has been chosen because it is a recommended open source true lossless format.

Write vector-based output. By using the VTK API and the vtkGL2PSExporter class, it becomes possible to save rendered objects in a high quality vector PostScript (PS/EPS) or PDF file. This class uses the GL2PS API to translate the OpenGL scene to vector format. It has some limitations since the PostScript is not an ideal language to represent complex 3D scenes but you can generate high quality vector PostScript with simple 3D scenes and most 2D plots. Thus, with a simple key press, the VTK_Mapper application offers the user, generation of a PDF file.

Run in batch and off screen mode. By using the VTK and Mesa libraries, it becomes possible to render a OpenGL scene in memory, without using hardware capacities of a graphic card. Thus, the VTK_Mapper application can produce a PDF file and a PNG file without any open window on your display or Xserver running. This feature effectively enables you to work off-line in a batch-oriented environment.

Automate mass-production documents. By using Python scripts and calls to the VTK_Mapper application expressed as a single line commands, you can mass produce documents. All interactive actions of the VTK_Mapper application can be retrieved as options in a UNIX-like command.

Use a high level interface. By using the Qt API, it has been possible to develop a very high level user interface. Many controls are possible with use of graphical and powerful widgets. It include a color control dialog, grid text layout to present the different variables and their attributes from the netCDF file loaded, control sliders, file selection dialogs and many others.

Process data. By using the COCO Python extension to CDMS API, it will be possible to process data easily. This feature is for now in a beta stage since there a small incompatibility with COCO and the CDMS generic grid structures. For now, processing is made by a simple evaluation of a Python expression.

1.8.3 Examples of use

The figures numbered 1.11 to 1.16 show screen shots obtained from VTK_Mapper and also the commands used to generate the screen shots from the command line.

In the examples, all the file arguments can be either a local file or a remote file served, for example, from the IPSL OPeNDAP/DODS server.

Download files from:
http://dods.ipsl.jussieu.fr/prism/gridsCF

and access the following:
http://dods.ipsl.jussieu.fr/fast/atlas/2L27_SE_2030_2039_output/

to find example files used.

Alternatively, it is be much easier to use the remote access capability by using syntax of the following form:

http://dods.ipsl.jussieu.fr/cgi-bin/nph-dods/dir1/dir2/file.nc

An example of the alternative ways of referencing the data is given in Figure 1.11
$ mapper.py -v -p orthographic -x 2L27_SE_2030_2039_histmth.nc tsol
$ mapper.py -v -p orthographic -x sampleCurveGrid4.nc sample

With remote access, this would read:
$ mapper.py -v -p orthographic -x
http://dods.ipsl.jussieu.fr/cgi-bin/nph-dods/...
...prism/gridsCF/sampleCurveGrid4.nc sample
$ mapper.py -l 100:2500:100 -v -p orthographic -x sampleGenGrid3.nc sample

Figure 1.11: Different model grids: rectilinear, curvilinear, generic
Figure 1.12: Switching between representation mode: cellsbounds, isocontours in a linear projection with an atmospheric model output (rectilinear grid).

Figure 1.13: Application of an operation to an atmospheric variable in an isocontour mode in a linear projection.
$ mapper.py -v --levels -2:18:1 --lindex 9 --color color3.sty
--gridfile IPSL.ORCA2_gridCF.nc --projection orthographic
2L27_SE_2030_2039_grid_T.nc votemper

If remote access is used, this example becomes:

$ mapper.py -v --levels -2:18:1 --lindex 9 --color color3.sty
--gridfile
http://dods.ipsl.jussieu.fr/cgi-bin/nph-dods/...
...prism/gridsCF/IPSL.ORCA2_gridCF.nc
--projection orthographic
http://dods.ipsl.jussieu.fr/cgi-bin/nph-dods/...
...fast/atlas/2L27_SE_2030_2039_output/2L27_SE_2030_2039_grid_T.nc
votemper

Figure 1.14: Switching between representation mode: cellsbounds, isocontours in an orthographic projection with an ocean model output (curvilinear grid).
$ mapper.py -x --actor isofill --camera camera4.sty --color color4.sty
--continents --continents_color 0.0,0.0,1.0 --continents_file
./polydouble_earth_continents.nc --continents_width 2 --equator
--equator_color 0.0,0.0,0.0 --equator_width 2 --grid --grid_color
0.45,0.45,0.45 --grid_delta 10 --grid_width 1 --levels_nb 15
--operation 'var-273.15' 2L27_SE_2030_2039_histmth.nc t2m

Figure 1.15: The Qt user interface and the corresponding command to re-produce the map.

$ mapper.py -v --offscreen -l 100:1000:50 --equator --continents
2L27_SE_2030_2039_histmth.nc rhum

Figure 1.16: An off screen call to produce a pdf file and a png file with a cells visualization mode.
1.8.4 Usage documentation

Usage: mapper.py [-h]
[-p projection] [-a actor]
[-n levels_nb] [-l min:max:delta]
[--bg r,g,b] [--fg r,g,b]
[--camera camera_object_file] [--color color_object_file]
[--index index] [--lindex index]
[--continents] [--continents_file file] [--continents_color r,g,b]
[--continents_width width]
[--boundaries] [--boundaries_color r,g,b]
[--boundaries_width width]
[--equator] [--equator_color r,g,b] [--equator_width width]
[--grid] [--grid_color r,g,b] [--grid_width width]
[--grid_delta delta] [--verbose] [--prefix prefixfilename]
[--offscreen] [-x] [--interface]
[--gridfile gridCF_file]
  var_file var

Options:
-h, -?, --help, -help
  Print this manual
-x, --interface
  Run the application with the GUI interface
-p, --projection
  Projection to choose in (linear,orthographic)
-a, --actor
  Actor to choose in : isofill, cell, cellbounds, isoline1, isoline2
  Other accepted syntax are: isofilled, cells, cellsbounds, isolines1,
  isolines2
-v, --verbose
  Verbose mode
-n, --levels_nb
  Number of levels should be in [3:100]
-l, --levels
  Levels expressed as minimum:maximum:delta
  Example: -l 2:32:4 from 2 to 32 by step of 4
  -l 0:0:4 from min to max by step of 4
--bg, --background
  Background color expressed as red, green, blue values in [0:1]
  Example: --bg 0.3,0.3,0.3
--fg, --foreground
  Foreground color expressed as red, green, blue values in [0:1]
  Example: --fg 1.0,1.0,1.0
--op, --operation
  Operation to apply on variable (use quote)
  Example: X--op ‘var*86400’
--op ‘(var*100)+273.15’
--camera
  Camera object file
--color
Color object file
--kindex
   Index for the 3rd dimension (vertical axis) of the variable to plot [1:n]
--lindex
   Index for the 4th dimension (time axis) of the variable to plot [1:n]
--boundaries, --continents, --equator, --grid
   Drawn if this option is present
--boundaries_color, --continents_color, --equator_color, --grid_color
   Color expressed as red, green, blue values in [0:1]
   Example: --boundaries_color 0.,0.,0.3
--boundaries_width, --continents_width, --equator_width, --grid_width
   Lines width expressed in [1:5]
--continents_file
   NetCDF continents file (CONT_LON,CONT_LAT variables)
--grid_delta
   Delta for grid lines (default=30)
--prefix
   Filename prefix used when PNG and PDF file are saved
   (default=picture)
--gridfile
   NetCDF file at the CF convention from where the mesh is read.
   If present, the "mask" variable is read and used in combination
   with the mask deduced from the variable.
   If gridfile not present, use only self descriptions of the variable.
--ratioxy
   Set the ratio between height and width
   for linear projection (default=1.0)
--offscreen
   Produce a PNG and a PDF file in an offscreen mode

1.8.5 Installation

You can download all the material from
http://dods.ipsl.jussieu.fr/vtk/VTK_Mapper
and follow the instructions you will find there.

1.8.6 Conclusion and Future Work

The initial set of defined goals has been covered. This confirms the strong advantages to build applications over the explored software architecture. Designing and implementing applications efforts will be continued in this way.

It has to be noted that the open source Paraview will be an excellent confirmation for the use of VTK since Paraview also uses this toolkit as the data processing and rendering engine and is a major keystone for large visualization projects.

For information, the open source project Paraview is presented as follows:
"Paraview is an application designed with the need to visualize large data sets in mind. The goals of the ParaView project include the following:

- Develop an open-source, multi-platform visualization application.
• Support distributed computation models to process large data sets.
• Create an open, flexible, and intuitive user interface.
• Develop an extensible architecture based on open standards.

ParaView runs on distributed and shared memory parallel as well as single processor systems and has been successfully tested on Windows, Linux and various Unix workstations and clusters. Under the hood, ParaView uses the Visualization Toolkit as the data processing and rendering engine.

ParaView is being developed by Kitware in conjunction with the Advanced Computing Laboratory at Los Alamos National Laboratory. ParaView is funded by the US Department of Energy ASCI Views program as part of a three-year contract awarded to Kitware, Inc. by a consortium of three National Labs - Los Alamos, Sandia, and Livermore. The goal of the project is to develop scalable parallel processing tools with an emphasis on distributed memory implementations.”

1.8.7 Useful On-line References

Links to related materials mentioned:

• VTK_Mapper application  
  http://dods.ipsl.jussieu.fr/vtk/VTKMapper
• VTK toolkit  
  http://www.vtk.org
• CDAT/CDMS (Climate Data Management System)  
  http://esg.llnl.gov/cdat
• COCO (CDMS overloaded for CF Objects)  
  http://prism.enes.org/WPs/WP4a/ProcessingLib
• Mesa for off screen rendering  
  http://www.mesa3d.org/
• PyQt (Python bindings for Qt) for the interface of the application  
  http://www.trolltech.com/qt  
  http://www.riverbankcomputing.co.uk/pyqt
• Paraview  
  http://www.paraview.org

1.9 Summary

Tools and techniques for processing and visualizing climate model data have been reviewed, extended and developed to prove that they can meet the wide range of requirements of climate modellers. Demonstration applications have been created that prove that data can be stored, read, processed and displayed in a variety of ways using the chosen tools.

The diversity of requirements has made it necessary to explore different visualization packages for the low end and for the high end visualization systems: VCS has been used in the low end system because it has a Python interface that enables it to be integrated very effectively with the chosen processing package, COCO. OpenDX has been extended for use in the high end because of its mature visual programming environment. A third package, VTK, has been reviewed because it has sufficient functionality to meet high end needs as well as having a Python interface that means that it could be incorporated effectively into low end systems.

In recognising the risks of using different packages for different purposes, development of methods to unite packages has been done: a colour table tool has been developed that will generate identical colour
tables for all three visualization packages which means data visualized using both low and high end applications can share the same colour scales. The ParaGen system has been developed that provides a way of describing and documenting script interfaces in an application-independent way. Use of both of these tools is strongly encouraged.

Plans for future development of these packages have been identified in the relevant sections. In particular, there is scope to extend the low end system so that the interface of low end scripts is represented within the PRISM user interface, and so that low end plots can be selected by the user in a flexible way. In general, each institution will have particular requirements for the processing of PRISM data, but the tools that have been examined provide a good basis from which to start.

1.10 Acknowledgements

The work for PRISM processing and visualisation was undertaken by the following people:
Patrick Brockmann (IPSL), Gabriella de Martino (KNMI), Camiel Severijns (KNMI), Frank Toussaint (MPI-M&D), Karin Meier-Fleischer (MPI-M&D), Michael Boettinger (DKRZ), Steve Mullerworth (Met Office), Mick Carter (Met Office), Rosalyn Hatcher (Met Office), Peter van Velthoven (KNMI), Reiner Vogelsang (SGI), Joerg Wegner (MPI-M&D), Jens Trapp (Sun), Thomas Schoenemeyer (NEC).