Development of a Climate Data Analysis Tool (CDAT)

Susan Mary Marlais

September 1997

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This report has been reproduced
directly from the best available copy.

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831
Prices available from (423) 576-8401

Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.,
Springfield, VA 22161

Development of a Climate Data Analysis Tool (CDAT)

Susan Mary Marlais

Manuscript date: September 1997

DEVELOPMENT OF A CLIMATE DATA
ANALYSIS TOOL (CDAT)

A Project
Presented
to the Faculty of
California State University, Chico

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Susan Mary Marlais
Fall 1997

DEVELOPMENT OF A CLIMATE DATA
ANALYSIS TOOL (CDAT)

A Project

by

Susan Mary Marlais

Fall 1997

APPROVED BY THE DEAN OF THE GRADUATE SCHOOL
GRADUATE, INTERNATIONAL, AND SPONSORED PROGRAMS:

Robert M. Jackson, Ph.D.

APPROVED BY THE GRADUATE ADVISORY COMMITTEE:

James Murphy, Ph.D., Chair

Kent Wooldridge, Ph.D.

DEDICATION

I would like to dedicate this thesis to my entire family, especially to my parents Andrew and Madaline Marlais, and my brother and sisters, Michael, Cathy, and Georgia. Thank you all for your support, love, and constant encouragement.

This thesis is also dedicated to all of my friends, you are a great bunch of people. I appreciate all of your friendships, and thanks for being there for me.

Finally, I would like to dedicate this thesis to the memory of two very special people, my mother Evelyn Marlais, and my friend Maggie Yatabe.

I LOVE YOU ALL!

ACKNOWLEDGMENTS

I would like to acknowledge and thank Dean Williams for his contributions to CDAT with the VCS module, command line interface, and all of his help and support throughout the development of CDAT. Most of all I would like to thank him for his friendship.

I would also like to acknowledge the entire staff of PCMDI for their support. I would especially like to thank Tom Phillips for all his comments and suggestions. Also, I would like to thank Karl Taylor and Clyde Dease for their help getting the regridding routine into CDAT. I would also like to thank Jerry Potter for his continued support towards the completion of CDAT.

Finally, I would like to thank the California State University Chico for this opportunity, especially Dr. Murphy and Dr. Wooldridge for being on my thesis committee.

TABLE OF CONTENTS

	PAGE
Dedication.....	iii
Acknowledgments	iv
List of Tables	vii
List of Figures.....	viii
Abstract.....	x
 CHAPTER	
I. Introduction.....	1
Background.....	1
Statement of the Problem.....	2
Purpose of the Project.....	2
Groups and Individuals Involved.....	3
Review of Related Work	3
Visualization and Computation System (VCS)	3
Grid Analysis and Display System (GrADS).....	4
NCAR Graphics.....	5
netCDF Calculator (nccalc)	6
II. Methodology	7
Selected Approach.....	7
CDAT Requirements	8
CDAT Extensions.....	13
Input to CDAT.....	13
Output from CDAT.....	13
Functions.....	16
Graphics.....	19

CHAPTER	PAGE
III. Experimental Work.....	21
Cdunif Module.....	21
LATS Module.....	26
VCS Module.....	29
PCMDI Module.....	33
CDAT Scripting.....	37
IV. Summary, Conclusions, and Recommendations.....	39
Summary.....	39
Conclusions.....	39
Recommendations.....	39
References Cited.....	41
Appendices	
A. List of Acronyms.....	43
B. List of CDAT Functions.....	45
C. Example Scripts.....	51
D. CDAT GUI.....	55

LIST OF TABLES

TABLE	PAGE
1. CDAT Data Type Definitions.....	9
2. CDAT Data Set Object Specifications.....	9
3. CDAT Cdunif Data Objects.....	10
4. CDAT LATS Data Objects	11
5. CDAT VCS Data Objects.....	11
6. CDAT Operators.....	12
7. CDAT Availability.....	12
8. CDAT Cdunif Commands.....	14
9. CDAT LATS Commands	15
10. CDAT Functions.....	16
11. CDAT PCMDI Functions	17
12. Array Functions.....	18
13. Discrete Fourier Transforms.....	18
14. Linear Algebra.....	19
15. CDAT VCS Commands.....	20

LIST OF FIGURES

FIGURE		PAGE
1.	Climate Data Analysis Tool (CDAT) Design Layout	8
2.	Cdunif Example of the open Function	22
3.	Cdunif Examples of the listdim, listvar, and listatt Functions	23
4.	Cdunif Example of the describe Function	23
5.	Cdunif Examples of the defvar and setdim Functions	24
6.	Cdunif Examples of the getatt, dimarray, dimlist, and getslab Functions.....	26
7.	LATS Examples fo the fldescr, convention, calendar, frequency, delta, and latscreate Functions.....	27
8.	LATS Examples of the year, month, day, hour, grid, vertical, time, latswrite, and latsclose Functions	29
9.	VCS Examples of the init, list, set, and page Functions.....	30
10.	VCS Examples of the plot, grid, resetgrid, clear, and close Functions.....	31
11.	VCS Examples of the animationgui, colormapgui, and colormap Functions.....	32
12.	VCS Examples of the cgm, raster, and script Functions.....	33
13.	PCMDI Examples of the cd, pwd, and ls Functions	34
14.	PCMDI Example of the slabinfo Function.....	35
15.	PCMDI Examples of the maskin and regrid Functions.....	36
16.	PCMDI Examples of the writenetcdf, writehdf, and writedrs Functions.....	36

FIGURE	PAGE
17. CDAT Script Example	37
18. Python Script Example	38

ABSTRACT

DEVELOPMENT OF A CLIMATE DATA ANALYSIS TOOL (CDAT)

by

Susan Mary Marlais

Master of Science in Computer Science

California State University, Chico

Fall 1997

The Climate Data Analysis Tool (CDAT) is designed to provide the Program for Climate Model Diagnosis and Intercomparison (PCMDI) at Lawrence Livermore National Laboratory, California, with the capabilities needed to analyze model data with little effort on the part of the scientist, while performing complex mathematical calculations, and graphically displaying the results. This computer software will meet the demanding needs of climate scientists by providing the necessary tools to diagnose, validate, and intercompare large observational and global climate model datasets.

This paper discusses some of the work already done in climate applications, with the emphasis on ingesting, manipulating, and graphically displaying climate data. It also describes the different problems that scientists encounter during their research, and how CDAT will address these problems.

Currently, the model data are output in many different file formats, forcing each scientist to convert the data to a familiar format. This conversion of the data is an invitation to introduce errors. By providing the means to ingest many different file formats, CDAT will eliminate the need for data conversion.

Once the data have been successfully ingested, scientists need to perform auxiliary calculations. These calculations have evolved over time with many duplications of software. CDAT will perform these calculations in a standard package, thereby reducing the time scientists spend writing their own routines to process the data.

Climate scientists typically use one software package to perform data calculations and another software package to graphically display the results. CDAT will allow the user to perform calculations on the data, and then send the results to a file that can be viewed as an image or a collection of images in an animation.

CDAT thus will provide climate scientists with an easy and fast method to ingest, manipulate, and graphically display climate data in an integrated fashion.

CHAPTER I

INTRODUCTION

Background

The Program for Climate Model Diagnosis and Intercomparison (PCMDI) at the Lawrence Livermore National Laboratory (LLNL) is coordinating the Atmospheric Model Intercomparison Project (AMIP), an international collaboration whose goal is to reduce the errors in model simulations of climate data. In this project, more than thirty climate modeling groups have submitted their data to PCMDI. With the AMIP experiment, PCMDI hopes to improve the methods for diagnosis, validation, and intercomparison of climate model data. In order to attain these goals, improved software for the analysis of climate data needs to be developed.

Currently, climate scientists have a wide choice of tools by which to analyze their climate data. For example, available computational and graphics packages include: the Visualization and Computation System (VCS), developed by PCMDI^[4]; the Grid Analysis and Display System (GrADS), developed by the Center for Ocean-Land-Atmosphere Studies (COLA)^[5]; the NCAR Graphics, developed by the National Center for Atmospheric Research (NCAR)^[6]; and the Network Common Data Format (netCDF) Calculator, developed by Pacific Northwest Laboratories^[7].

Statement of the Problem

The problem facing climate scientists is not the absence of software to analyze their data; it is, rather, a shortage of diagnostic tools that are consistent, flexible, portable, adaptable, efficient, and easy to use. As a consequence, many scientists are writing their own programs to ingest and manipulate their data. Redundancy in such efforts diverts time that otherwise would be spent on research to the debugging and enhancing of these programs. The resulting software is often not user-friendly or portable, and does not promote common standards within the climate community.

Another obstacle to sharing analysis software is the wide variety of data file formats that are in use, making it necessary to write programs to convert data to one's preferred file format conventions. This data conversion requires additional expenditure of effort on testing and quality assurance. Software to perform such tasks transparently would thus be a valuable asset for climate scientists, allowing them to spend their time analyzing, rather than processing, data.

Purpose of the Project

The purpose of this project is to develop the Climate Data Analysis Tool (CDAT), which will run as a standalone process or within PCMDI's Visualization and Computation System (VCS). CDAT will provide climate scientists with an easy and fast method to read different file formats and to

analyze their data. It will include a set of pre-defined functions to allow the user to manipulate the data and send the output to a file which can be viewed as an image, or as a collection of images in an animation. CDAT will have a gradual learning curve, allowing the novice user to quickly obtain useful results.

Groups and Individuals Involved

The Program for Climate Model Diagnosis and Intercomparison (PCMDI) at Lawrence Livermore National Laboratory and The National Center for Atmospheric Research (NCAR) collaborated on this project. Susan Peterson and Dean Williams from PCMDI developed the code, while Jim Boyle, Bob Drach, Jerry Potter, Ben Santer, and Karl Taylor from PCMDI, and Jim Hack from NCAR provided advice or software that was used to develop CDAT.

Review of Related Work

Visualization and Computation System (VCS)

VCS was developed by a team of computer scientists at the Lawrence Livermore National Laboratory for PCMDI's climate scientists to use in their research, and to diagnose and validate climate model data^[4]. VCS can be used to plot, animate, or compute data fields. The data displayed in VCS can be saved as a Computer Graphics Metafile (CGM) file, raster file, and/or PostScript file.

In the VCS model^[4], displays are produced by defining a trio of attribute sets called "primary elements". These include the "data", which defines what to display, the "template", which defines where to display the data and the

“graphics method”, which defines how to display the data. VCS currently supports different map projections such as Cartesian, Mollweide, and polar stereographic and different graphics methods such as isoline, isofill, boxfill, x-y, and scatter plots.

VCS can ingest data from several different file formats. These include the Data Retrieval and Storage (DRS) format, developed by PCMDI; the Grid Analysis and Display System (GrADS) format, developed by the Center for Ocean-Land-Atmosphere Studies^[5]; the Hierarchical Data Format (HDF), developed by the National Center for Supercomputing Applications^[12]; and the Network Common Data Format (netCDF), developed by the Unidata program for the National Science Foundation Division of the Atmospheric Sciences^[11].

VCS is written in ANSI C and uses the X Graphical Kernel System (XGKS)^[8] for its graphics library and X/Motif^[9] for its window interface. The software binaries are freely available on all major UNIX platforms on the World Wide Web at URL <http://www-pcmdi.llnl.gov/software/vcs>.

VCS is a very effective graphics package, but it does have its limitations. Currently, VCS only implements pre-defined functions which are somewhat limited. Also, it can only display one- and two-dimensional graphics methods. VCS has limited scripting capabilities and no command line interface.

Grid Analysis and Display System (GrADS)

The Grid Analysis and Display System (GrADS) is an interactive tool that is currently being used for the analysis and display of atmospheric data^[5].

GrADS is available on most UNIX platforms and on DOS-based PCs. GrADS is freely available via anonymous ftp at grads.iges.org.

GrADS uses a 4-dimensional data model, where the dimensions are latitude, longitude, level, and time. The data attributes are described in a metafile, and both gridded and station data may be described. The data may be either binary or GRIdded Binary (GRIB) formats. Gridded data may be non-linearly spaced while Gaussian grids are directly supported by GrADS. A Gaussian grid is a non-linearly spaced grid.

Operations on the data may be performed directly, interactively, or by entering FORTRAN-like expressions on the command line. There is a set of built-in functions provided in the software package. In addition, users may add their own functions as external routines in any language.

After the data have been accessed and manipulated, they can be plotted using a graphical display method. These methods include: line, bar, scatter plot, contour, wind vector, and grid box. The graphics may also be output in PostScript format for printing.

GrADS also provides an interpreted scripting language. A script may display widgets and/or graphics. The scripting language can also be used to automate complex multi-step calculations or displays, and to run GrADS in batch mode.

GrADS is limited in that it does not allow the user to create animations for viewing data. Even though GrADS is a powerful tool, some scientists are

hesitant to use it because it is difficult to learn. GrADS also has a very limited widget set.

NCAR Graphics

The NCAR Graphics package has been an ongoing project since the early 1960s^[6]. It is composed of a collection of low-level FORTRAN routines for drawing simple graphical elements. NCAR Graphics has developed over the years to include higher-level graphics utilities based on the XGKS^[8] standard. All utilities can be accessed through C or FORTRAN calls.

Currently, NCAR Graphics is available on the following UNIX platforms: DEC running OSF/1 2.0 or Ultrix 4.3, Cray Y-MP running UNICOS 8.0.2, Hewlett Packard HP9000 Series 700 running HP-UX 9.01, IBM RISC System/6000 running AIX 3.2, SGI running IRIX 5.2, and Sun-4 series running SunOS 4.1 or Solaris 2.3.^[6]

The NCAR Graphics package can be difficult to use because it does not have a consistent user interface. This is because the different utilities were developed by several people over the years. Its components have been collected into one package, but new users can be confused by the differences in the calls needed to control the utilities. The lack of animation routines is also a limitation. NCAR Graphics is not distributed freely: it costs \$4000.00 per license, which limits its use in the research field.

netCDF Calculator (nccalc)

The netCDF Calculator (nccalc) is based on the EPIC Data System data files^[7]. These data files can be in any format that is supported by EPIC, for example binary or netCDF. The nccalc can handle files up to four dimensions, with the coordinates being (x,y,z,t) that is, (longitude, latitude, level, time). The dimensions can also be given as indices (i,j,k,l).

This calculator has its own language. The data types included are scalar, string, slab, and field. A scalar is defined as a number, or a representation of a number; it can also be an array. A string is a character string or a variable. A slab is defined as a region represented by (x,y,z,t) values. A field is like a slab, except it represents a range for each of the (x,y,z,t) coordinates.^[7]

The netCDF calculator has the basic arithmetic operations and functions, although it does not include any vector functions. The user has limited access to different data formats. It is also not accessed through a Graphical User Interface (GUI), has limited graphics, and does not have any animation routines.

CHAPTER II

METHODOLOGY

Selected Approach

To develop CDAT we used Python^[1], an interpreted, object-oriented scripting language, whose strengths include very clear syntax and powerful data structures. Python was developed by Guido van Rossum, from the Corporation for National Research Initiatives (CNRI). Python can be extended through modules written in C or C++ that give access to most UNIX library functions, and to packages such as X11/Motif^[9]. Python users have developed modules for image processing, graphical user interfaces, World Wide Web programming, and numerical extensions.^[1] Python has several built-in functions, including basic mathematical operations, that can be applied without use of an import statement. There are also extensions that can be imported into Python, giving the language more functionality.

CDAT consists of several modules written in C that work through Python to give the tool all its capabilities and flexibility. The completion of this project required the development of several modules to perform data input and output. Some of these modules were already in place, so that only the interface between the modules and Python needed to be developed.

CDAT Requirements

The development of CDAT required software that was already in use by PCMDI and NCAR, as well as the development of new software packages. Python, the centralized control module, allows the different packages to interact with each other. Figure 1 shows the design layout for CDAT. The design includes modules for input and output of data, graphics, and numerical routines defined in the different function modules. The design also includes a command line interface and a graphical user interface for entering commands into CDAT.

Figure 1. Climate Data Analysis Tool (CDAT) Design Layout

The CDAT package requires the use of several different data types and data set objects, which are defined in Tables 1 and 2 respectively.

TABLE 1. CDAT Data Type Definitions

Type	Definition
byte	8-bit integer type
char	8-bit character type
short	16-bit integer type
int	32-bit integer type
long	64-bit integer type
float	32-bit float type
double	64-bit float type
string	arrays of characters

TABLE 2. CDAT Data Set Object Specifications

Type	Definition
dimension	specifies the shape of one or more of the multidimensional variables contained in a file
attribute	a distinctive feature associated with a dimension or variable
slab	a multidimensional object that specifies shape of its dimensions, attributes, and contains data
array	a multidimensional object
vector	a one-dimensional array
scalar	a single valued dimension
region	range of data (geometry)
file	contains dimensions, dimensions attributes, variables, variable attributes, and data

Each module used to implement CDAT contains different objects.

The CDAT cdunif data objects are defined in Table 3.

TABLE 3. CDAT Cdunif Data Objects

Type	Definition
file	filename - name of the file fid - id number for the file ndims - number of dimensions nvars - number of variables natts - number of attributes recdim - dimension number of unlimited variable defaultname - default variable for opened file
slab	name - name of the variable filename - name of file datatype - type of data in the file ndims - number of dimensions dims[] - array of dimension ids natts - number of attributes atts[] - linked list of attributes data[] - array for the actual data missing - missing data value
attribute	name - name of attribute datatype - type of attribute len - number of elements in attribute values - attribute value
dimension	name - name of dimension units - units of the dimension len - length of the dimension datatype - type of dimension bounds - coordinate boundaries or limits weights - coordinate area weights cycle - cycle values, used for wrapping order - order of dimensions start - starting dimension value end - ending dimension value values - coordinate values

The CDAT LATS data objects are defined in Table 4. The LATS module also uses the cdunif slab object.

TABLE 4. CDAT LATS Data Objects

Type	Definition
file	filename - name of the file latsid - id number for the file filetype - flag to determine what type of file is created comments - file comments center - organization which produced the data model - model which produced the data timeunits - time units written to file delta - number of units in time increment nslab - number of slabs in the list nvert - number of vertical dimensions defined calendar - calendar type convention - convention for writing data/metadata frequency - file time frequency slablist - pointer to list of file slabs verlist - vertical dimension list
slab	the cdunif slab data object

The CDAT VCS data objects are defined in Table 5. The VCS module also uses the cdunif slab object.

TABLE 5. CDAT VCS Data Objects

Type	Definition
canvas	VCS drawing window
template	collection of objects which describes where to display the slab
graphics	collection of objects which describes how to display the slab
slab	the cdunif slab data object
colormap	contains 240 user definable colors
fill area	hatch styles and colors
line	line type, width, and color
list	pairs of numerical and character values
marker	marker type, size, and color
text	font type, spacing, expansion, and color

text orientation	height, angle, path, and alignment
animation	a sequence of raster images viewed consecutively

CDAT also contains the standard mathematical and logical operators, listed in Table 6.

TABLE 6. CDAT Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
%	modulus (return the remainder after division)
^	power
=	equals, assigned
==	equal to
!=	not equal
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
!	not, boolean operation that returns the logical inverse of the operand
&&	logical and, a boolean operation that returns true if both arguments are true
	logical or, a boolean operation that returns true if one of the two arguments are true

The following CDAT operators work with missing value representations: +, -, *, /, %, and ^. This is very important in the climate community, since some of the data being analyzed may contain missing values.

Due to the anticipated large number of users, CDAT will need to be ported to several different platforms, shown in Table 7.

TABLE 7. CDAT Availability

Platform	Operating System
Cray C90 and J90	Unicos 8.0
Digital Equipment Corporation (DEC)	OSF 3.2
Hewlett Packard (HP)	HP-UX 9.0.5 or higher
International Business Machines (IBM)	AIX 3.5 or higher
Personal Computer (PC)	Linux 3.0.3 or Solaris
Silicon Graphics, Inc. (SGI)	IRIX 5.2, 5.3, or 6.1
Sun Microsystems	SunOS 4.1.3, Solaris 2.4

CDAT Extensions

Input to CDAT

A command line interface has been provided, thus allowing arguments to be used to input commands into CDAT. The user accesses the command line interface by typing “cdat”. The commands entered are sent through Python to the different extensions. Another means of sending commands to Python is through the GUI, which is accessed by typing “cdatgui”, it will also be accessible from within VCS.

PCMDI's Common Data UNiform InterFace (cdunif) is used to input data into CDAT in the following file formats: Data Retrieval and Storage (DRS), Grid Analysis and Display System (GrADS), GRIdded Binary (GRIB), Hierarchical Data Format (HDF), and Network Common Data Format (netCDF). An interface between cdunif and Python has been developed, giving the CDAT user the capabilities of cdunif. Table 8 describes the CDAT cdunif commands. In Table 8 the <> convention is used to indicate the different parameters that can be passed to the function call.

When the Climate Data Management System (CDMS), also under development at PCMDI, is released, a module will be developed to interface CDMS to Python. The CDMS module will be yet another method for ingesting data into CDAT, but it will allow CDAT to automatically locate and extract physics quantities, such as variables, dimensions, and grids.

Output from CDAT

To output data from CDAT, one may use the Library of AMIP data Transmission Standards (LATS), a software library developed by PCMDI.

LATS outputs

TABLE 8. CDAT Cdunif Commands

Command	Definition
<code>f = cu.open('file')</code>	opens a <i>file</i> ; returns a python file object
<code>f.listdim()</code>	returns a list of dimensions associated with file object
<code>f.listvar()</code>	returns a list of variables associated with file object
<code>f.listatt('varname')</code>	returns a list of attributes associated with file object, for <i>varname</i>
<code>att = f.getatt('var','attribute')</code>	returns an <i>attribute</i> for <i>var</i> associated with file object
<code>f.defvar('varname')</code>	sets the default variable for a file object to <i>varname</i>
<code>f.describe('varname',<'all'>)</code>	returns a description of a variable, if no <i>varname</i> is passed, it will use the default variable, if <i>all</i> is passed, it returns a more complete description
<code>dim1 = cu.setdim(start, end)</code>	sets dimension values for a dimension object with world coordinates, where <i>start</i> & <i>end</i> are defined as double
<code>dim=f.dimlist('dimname')</code>	returns a list of dimension values associated with file object for <i>dimname</i>

<code>dim=f.dimarray('dimname')</code>	returns an array of dimension values associated with file object for <i>dimname</i>
<code>slab = f.getslab('var')</code> <code>slab = f.getslab('var',dim1,dim2,...)</code> <code>slab = f.getslab('var',-180,180,-90,90,1,120)</code> <code>slab = f.getslab('var', ':', ':', 1,12)</code>	returns a slab of data, where <i>var</i> is the variable that will be returned in slab; this call is used for world coordinates, where the ':' and '*' represent a wild card and the values in the file are used

global, rectangular gridded data. After a LATS file has been defined and created, a single function call to the LATS module will write out data in a horizontal-latitude slice of a variable. Data from LATS can be output in either GRIB or netCDF file formats. Table 9 defines the CDAT LATS output commands.

A user can also generate output without using LATS, writing out data in ASCII, DRS, HDF or netCDF formats, which are less restrictive conventions than that from LATS.

TABLE 9. CDAT LATS Commands

Command	Definition
<code>lats.fldescr('center','model','comments')</code>	defines the center, model and comments for the file
<code>lats.convention('conv')</code>	defines the convention of the data to be written to the file
<code>lats.calendar('cal')</code>	defines the calendar type
<code>lats.frequency('freq')</code>	defines the frequency of the variables to be written to the file
<code>lats.delta(d)</code>	sets the delta or time increments to be used with the frequency
<code>f=lats.create('filename')</code>	creates a lats file with the given

	<i>filename</i>
<code>lats.year(<i>beg</i>,<i>end</i>)</code>	defines the years to be written to the file
<code>lats.month(<i>beg</i>,<i>end</i>)</code>	defines the months to be written to the file
<code>lats.day(<i>beg</i>,<i>end</i>)</code>	defines the days to be written to the file
<code>lats.hour(<i>beg</i>,<i>end</i>)</code>	defines the hours to be written to the file
<code>lats.grid('name','type')</code>	defines a LATS grid
<code>lats.vertical('name','type')</code>	defines a vertical dimension
<code>lats.time('name')</code>	sets the timestat of the data
<code>f.write(<i>slab</i>)</code>	writes the <i>slab</i> to a file that is open
<code>f.close()</code>	closes the lats file

A user can also output data in Common Gateway Interface (CGI)^[2] scripts. CGI is a standard for interfacing external applications with information servers, such as Hyper Text Transfer Protocol (HTTP)^[10] or World Wide Web servers. A CGI program, is executed in real time, so that it can output dynamic information^[2], in contrast to the static Hyper Text Markup Language (HTML)^[10] text files that also can be retrieved from a Web server.

Functions

The user has access to the different numerical calculations that are supplied in CDAT through Python. The standard Python functions available in CDAT are defined in Table 10.

TABLE 10. CDAT Functions

Function	Definition
<code>abs(<i>x</i>)</code>	returns the absolute value of <i>x</i>
<code>cos(<i>x</i>)</code>	returns the cosine of <i>x</i>
<code>sin(<i>x</i>)</code>	returns the sine of <i>x</i>
<code>tan(<i>x</i>)</code>	returns the tangent of <i>x</i>

<code>acos(x)</code>	returns the arc cosine of x
<code>asin(x)</code>	returns the arc sine of x
<code>atan(x)</code>	returns the arc tangent of x
<code>cosh(x)</code>	returns the hyperbolic cosine of x
<code>sinh(x)</code>	returns the hyperbolic sine of x
<code>tanh(x)</code>	returns the hyperbolic tangent of x
<code>acosh(x)</code>	returns the arc hyperbolic cosine of x
<code>asinh(x)</code>	returns the arc hyperbolic sine of x
<code>atanh(x)</code>	returns the arc hyperbolic tangent of x
<code>exp(x)</code>	returns the exponential of x
<code>log(x)</code>	returns the logarithm of x
<code>log₁₀(x)</code>	returns the logarithm base 10 of x
<code>sqrt(x)</code>	returns the square root of x
<code>int(x)</code>	casts x to an integer
<code>float(x)</code>	casts x to a float
<code>long(x)</code>	casts x to a long

The CDAT external functions have been supplied by NCAR and PCMDI. Table 11 defines the PCMDI functions; the NCAR functions have not yet been implemented.

TABLE 11. CDAT PCMDI Functions

Function	Definition
<code>maskin('slab', 'mask')</code>	creates the input mask for <i>slab</i> , where <i>mask</i> can be either land or ocean
<code>new=regrid('slab1', 'grid1', 'slab2', 'grid2')</code>	regrids a <i>slab1</i> to the grid of <i>slab2</i> , where <i>grid1</i> and <i>grid2</i> can be either the model or the name of the grid
<code>writenetcdf('filename', ['mode'])</code>	creates a netCDF file, where <i>mode</i> = r (replace) or a (append)
<code>writehdf('filename', ['mode'])</code>	creates a netCDF file, where <i>mode</i> = r (replace) or a (append)
<code>writenetcdf('filename', ['mode'])</code>	creates a netCDF file, where <i>mode</i> = r (replace) or a (append)

<code>slabinfo(<i>slab</i>)</code>	returns information about <i>slab</i>
<code>pwd()</code>	returns the present working directory
<code>ls()</code>	returns a list of files and directories
<code>cd('dir')</code>	change directory to <i>dir</i>

A numeric module for Python, developed by Konrad Hinsén, and James Hugunin, extends the built-in math functions of Table 10 to include functions pertinent to array operations, discrete Fourier transforms, and linear algebra. These extensions, which are available to CDAT users are defined in Tables 12 - 14. The numeric routines have been modified to handle missing data, and to incorporate the attributes that PCMDI requires in the slab objects. Two new functions were also added to the existing numeric routines that allow the user to set the missing value and show the missing value for the data they are working with.

TABLE 12. Array Functions

Function	Definition
<code>array(sequence, typecode=None)</code>	creates an array object from the given <i>sequence</i>
<code>zeros(shape, typecode=Integer())</code>	creates an array object of <i>shape</i> and given <i>typecode</i> filled with zeros
<code>ones(shape, typecode=Integer())</code>	creates an array object of <i>shape</i> and given <i>typecode</i> filled with ones
<code>dot(a, b)</code>	returns the dot product of <i>a</i> & <i>b</i>
<code>min(x)</code>	returns the smallest item in a non-empty array <i>x</i>
<code>max(x)</code>	returns the largest item in a non-empty array <i>x</i>
<code>len(x)</code>	returns the length of <i>x</i>
<code>x.reverse()</code>	reverses the items in <i>x</i>
<code>clip(a, min, max)</code>	clips the values in <i>a</i> to lie between <i>min</i> & <i>max</i>

<code>transpose(a, axes)</code>	swaps the axes of <i>a</i> , given the new order <i>axes</i>
<code>sort(a, axis=-1)</code>	sorts the elements of <i>a</i> along the given <i>axis</i>
<code>indices(shape)</code>	returns an array of indices for the given shaped array
<code>concatenate((a1, a2, ..., an) axis=0)</code>	concatenates all arrays along the given axis
<code>add.reduce(a)</code>	returns a new array, reduced in dimensions

TABLE 13. Discrete Fourier Transforms

Function	Definition
<code>fft(a, n=None, axis=-1)</code>	returns the <i>n</i> point discrete Fourier transformation of <i>a</i> (<i>n</i> defaults to the length of <i>a</i>)
<code>inverse_fft(a, n=None, axis=-1)</code>	returns the <i>n</i> point inverse discrete Fourier transform of <i>a</i> (<i>n</i> defaults to the length of <i>a</i>)
<code>real_fft(a, n=None, axis=-1)</code>	returns the <i>n</i> point discrete Fourier transform of the real valued array <i>a</i> (<i>n</i> defaults to the length of <i>a</i>)
<code>ff2d(a, s=None, axes=(-2,-1))</code>	the 2d fft of <i>a</i>

TABLE 14. Linear Algebra

Function	Definition
<code>solve_linear_equations(a, b)</code>	solves linear equations given the matrices <i>a</i> & <i>b</i>
<code>inverse(a)</code>	returns the inverse of the matrix <i>a</i>
<code>eigenvalues(a)</code>	returns the eigenvalues for <i>a</i>
<code>eigenvectors(a)</code>	returns <i>u</i> , <i>v</i> where <i>u</i> is the eigenvalues and <i>v</i> is a matrix of eigenvectors
<code>single_value_decomposition(a)</code>	returns the singular value decomposition of the matrix <i>a</i>

<code>generalized_inverse(a)</code>	returns the generalized inverse of the matrix <i>a</i>
<code>determinate(a)</code>	returns the determinate of the square matrix <i>a</i>
<code>linear_least_squares(a, b, rcond=1.e-10)</code>	takes two arrays <i>a</i> & <i>b</i> and a small number; returns a 4-tuple (x, residuals, rank, s)

Graphics

CDAT uses VCS scripting for graphical display of the data. VCS is used to plot the slabs of data that are retrieved through the *getslab* call in the *cdunif* module. The VCS module also has routines to modify the display. The VCS commands that are available in CDAT are explained in Table 15.

TABLE 15. CDAT VCS Commands

Command	Purpose
<code>x = vcs.init()</code>	initializes the VCS module
<code>x.open()</code>	opens a vcs canvas object
<code>x.close()</code>	closes a vcs canvas object
<code>x.page()</code>	changes the orientation of the vcs canvas object, either portrait or landscape
<code>x.list('element')</code>	returns a list of elements depending on what it is passed, where <i>element</i> can be template, graphics, or data, etc.
<code>x.plot(slab)</code> <code>x.plot(slab, 'template', 'graphics')</code>	plots a slab with the defined <i>template</i> and <i>graphics</i> method
<code>x.grid(0,10,5,10)</code> <code>x.grid(':',':',3,3)</code>	sets the plotting region, if ':' or '*' are specified the coordinate values in the slab are used
<code>x.resetgrid()</code>	resets the display grid back to the default values
<code>x.set('template', 'type')</code>	sets the current default template or graphics method to the parameter that is passed
<code>x.clear()</code>	clears the canvas
<code>x.colormap('color table name')</code>	assigns one of the color table entries as the active color table
<code>x.script('script file')</code>	executes a VCS script
<code>x.raster('filename', ['mode'])</code>	creates a raster file that can be used in a animation <i>mode</i> = r (replace) or a (append)
<code>x.cgm('filename', ['mode'])</code>	creates a cgm file <i>mode</i> = r (replace) or a (append)
<code>colormapgui</code>	pops up the VCS colormap GUI
<code>animationgui</code>	pops up the VCS animation GUI

CHAPTER III

EXPERIMENTAL WORK

Cdunif Module

The cdunif module gives CDAT the ability to ingest data in several different file formats, see Chapter 2, Input to CDAT for details. This module also allows the user to retrieve information about the file and the physical quantities contained there in. The user can also define and retrieve a slab of data from the opened file, without knowing anything about the data properties. The user can also use the information from the file to retrieve data in specified coordinate values.

This module, the interface between Python and cdunif, was developed by creating a Python C extension type. The module extends Python with the routines written in the C programming language. The Python API is used to interact with C, passing arguments between routines, handling garbage collection, and references to all Python objects.

To open a file, the user types *f=cu.open('filename')*, where the chosen name *f* becomes an opened cdunif file object, and *filename* is the name of an existing file to be accessed including, as necessary, the complete path name.

The CDAT command to open a cdunif file is shown in Figure 2. In this, and all other figures illustrating CDAT commands the output from the command is displayed in bold font.

```
CDAT> f=cu.open('clt.nc')
The cdunif file clt.nc is now opened
```

Figure 2. Cdunif Example of the open Function

CDAT has several commands to obtain information about the opened files. The command *f.listdim()* is used to list the dimensions contained in an opened cdunif file, where *f* is an opened cdunif file object. A user can pass a variable to the *listdim* function and it will only return the dimensions associated with that variable for the opened cdunif file object. The user can list the variables in the opened cdunif file with *f.listvar()*, where *f* is an opened cdunif file object. The attributes of the dimensions and variables contained in the file can be listed with *f.listatt()*, to list all of the attributes in the file, or with *f.listatt('var')* for a specific variable or dimension, in both cases *f* is the opened cdunif file object. To get a more detailed description of a variable contained in the file, the command *f.describe('var')* or *f.describe('var', 'all')* can be used. This command gives the variables attributes and dimensions. Where *f* is an opened cdunif file object, *var* is the name of a variable contained in the file object *f*. The option *all* can be used to obtain a complete listing, including the dimension values.

The CDAT commands to list dimensions, variables, and attributes, and to describe the data contained in an opened cdunif file object are shown in Figure 3 and 4. The *listdim* example shows a listing of all dimensions contained in the file, not just for one variable.

```

CDAT> f.listdim()
***** Dimension List *****
The dimensions in the file clt.nc are:
time, longitude, latitude

CDAT> f.listvar('clt')
***** Variable List *****
The variables in the file clt.nc are:
clt

CDAT> f.listatt('clt')
***** Attribute List *****
The attributes in the file clt.nc for variable clt are:
comments, long_name, units, grid_name, grid_type

```

Figure 3. Cdatunif Examples of the listdim, listvar, and listatt Functions

If the CDAT describe command is implemented with the *all* option, for example, *f.describe('clt', 'all')*, the output will list the dimension values, not just the first and last values.

```

CDAT> f.describe('clt')
***** Variable Description *****
Name: clt
comments: YONU_AMIP1
long_name: Total cloudiness
units: %
grid_name: YONU4X5
grid_type: gaussian
time_statistic: average
Data Type: Float
** Dimension 1 **
Name: longitude
Units: degrees_east
Length: 72
First: -180.0000 Last: 175.0000
** Dimension 2 **
Name: latitude
Units: degrees_north
Length: 46
First: -90.0000 Last: 90.0000
** Dimension 3 **
Name: time
Units: months since 1979-1-1 0
Length: 120

```

First: 0.0000	Last: 119.0000
---------------	----------------

Figure 4. Cdunif Example of the describe Function

The CDAT user can also set default values that can be used repeatedly, rather than repeatedly typing in the values. To set a default variable for an opened cdunif file object the command *f.defvar('var')* is used. Where *f* is an opened cdunif file object and *var* is the variable that is to be used as the file objects default variable. When a default variable has been set for a file object the user can use the default variable for all commands that require a variable parameter instead of passing in a variable argument. A dimension can be set with the *dim1=cu.setdim(beg, end)* command. Where *dim* becomes the new dimension object, *beg* is the first dimension value and *end* is the last dimension value.

The CDAT commands to set the default variable for an opened cdunif file object, and to create a dimension object that can be used in the call to *getslab* are shown in Figure 5.

CDAT> f.defvar('clt') Default variable for clt.nc is now set to clt CDAT> d1=cu.setdim(-180,180) First dimension value is now set to -180 Last dimension value is now set to 180

Figure 5. Cdunif Examples of the defvar and setdim Functions

There are three commands to retrieve data from an opened cdunif file object. The first command, *attribute=f.getatt('var','attname')* is used to retrieve an attribute of a particular variable contained in the opened cdunif file object. The

command returns the attribute for the variable in a string object. The *var* parameter is the variable the user wants the attribute for and the *attname* parameter is the actual attribute the user wants to retrieve.

The next command used to retrieve data from an opened cdunif file object has two different forms, it can either return a Python array or a Python list. The command *dimension=f.dimarray('name')* is used to retrieve the dimension array from the file object, where *dimension* is the array object containing the dimension values, *f* is the opened cdunif file object and *name* is the name of the dimension that is being retrieved. The *dimension=f.dimlist('name')* is used to return the same information as the *dimarray*, but the command returns a list instead of an array.

The command to retrieve a slab of data from an opened cdunif file object has several arguments that can be passed into the command. When the command is used with this format *slab=f.getslab('var')*, where *slab* is the array object that is returned from the call to *getslab*, *f* is the opened cdunif file and *var* is the variable that is being retrieved from the file, the command returns all the values from the file as they are stored in the file. The CDAT user has the option of using wild cards in the call to *getslab* also. These wild cards are either a ':' or '*'. When they are used in the call to *getslab*, the default values from the file are returned. The wild cards are useful when the user wants to define certain dimensions and use the default values from the file for other dimensions. Another way the *getslab* command can be called is

$slab=f.getslab('var',beg1,end1,\dots,begn,endn)$, where the *beg* and *end* are the first and last values of the dimension. The dimension object that was returned by the call to *setdim* can be passed into the call to *getslab*. This call looks like,
 $slab=f.getslab('var',dim1,dim2,\dots,dimn)$. Figure 6 shows examples of the functions that retrieve data from a cdunif file object.

```

CDAT> grid=f.getatt('clt','grid_name')
CDAT> grid
'YONU4X5'

CDAT> lat=f.dimarray('latitude')
CDAT> lat
array([-90.0, -86.0, -82.0, -78.0, -74.0, -70.0, -66.0, -62.0, -58.0, -54.0, -50.0,
       -46.0, -42.0, -38.0, -34.0, -30.0, -26.0, -22.0, -18.0, -14.0, -10.0, -6.0, -2.0, 2.0,
        6.0, 10.0, 14.0, 18.0, 22.0, 26.0, 30.0, 34.0, 38.0, 42.0, 46.0, 50.0, 54.0, 58.0,
        62.0, 66.0, 70.0, 74.0, 78.0, 82.0, 86.0, 90.0], 'F')

CDAT> lat=f.dimlist('latitude')
CDAT> lat
[-90.0, -86.0, -82.0, -78.0, -74.0, -70.0, -66.0, -62.0, -58.0, -54.0, -50.0, -46.0,
 -42.0, -38.0, -34.0, -30.0, -26.0, -22.0, -18.0, -14.0, -10.0, -6.0, -2.0, 2.0, 6.0,
 10.0, 14.0, 18.0, 22.0, 26.0, 30.0, 34.0, 38.0, 42.0, 46.0, 50.0, 54.0, 58.0, 62.0,
 66.0, 70.0, 74.0, 78.0, 82.0, 86.0, 90.0]

CDAT> s=f.getslab('clt')
CDAT> s=f.getslab('clt','.',-50,50,':')
CDAT> s=f.getslab('clt',0,360,-50,50,1,12)
CDAT> s=f.getslab('clt', d1, d2)

```

Figure 6. Cdunif Examples of the getatt, dimarray, dimlist, and getslab Functions

LATS Module

The LATS module is the interface between Python and PCMDI's LATS software routines. Like the cdunif module, the LATS module was written in C and is a Python extension. The LATS module was designed to write out data in either netCDF or GRIB format, according to a standard structure for the AMIP project. This module allows the user to create a file and write variables into it.

To create a LATS file object, several commands must first be called.

The *fldescr*('center', 'model', 'comments') command is used to define the *center* (name of the modeling group that is the source of the data), *model* (name of the model version that created the data), and *comments* (description of the data).

To define the data writing convention the *convention('conv')* command is used, where *conv* is either PCMDI netCDF format, GrADS WMO GRIB format, or COARDS netCDF format observing the COARDS metadata standard. The *calendar('cal')* command is used to define the calendar type, which may be one of the following types: STANDARD, standard Gregorian calendar; JULIAN, with leap years being divisible by four; NOLEAP, 365 days/year (no leap years); 360, each year consists of 12 30-day months; CLIM, climatological time with 365 days (no associated year); CLIMLEAP, climatological time with 366 days; or CLIM360, climatological time with 360 days.

The *frequency('freq')* command is used to define the time frequency of variables to be written to the file. Options include: YEARLY, MONTHLY, COMP, WEEKLY, DAILY, HOURLY, or FIXED. To set the time increment delta for the time frequency, the *delta(d)* command is used, where *d* is the number of time units in the time increment, and units is specified by the frequency.

Once all of the options for the LATS file have been set the command *file=latscreate('latsfile')* can be run, where *file* is the created LATS file object, and *latsfile* is the actual name of the LATS file. Examples of the use of these LATS commands are shown in Figure 7.

```
CDAT> fldescr('BMRC','BMR2','AMIP II data set')
CDAT> convention('pcmdi')
CDAT> calendar('standard')
CDAT> frequency('monthly')
CDAT> delta(1)

CDAT> fl=latscreate('latsfile.nc')
```

The lats file object latsfile.nc has been created

Figure 7. LATS Examples of the fldescr, convention, calendar, frequency, delta, and latscreate Functions

After creation of the LATS file object, the user sets up the data to be written into the new file object. The *year(beg,end)* command is used to define the years that will be written to the file, where *beg* is the first year to be written, and *end* is the last year to be written to the file. The *month(beg,end)* command is used to set the first and last months that will be written to the file for each year that is being written. The *day(beg,end)* command is used to set the first and last days that will be written into the file for each month that was written. The *hour(beg,end)* command sets the hours that are to be written in the file for each day that was written. The frequency defines which elements of time will be used in the writing of the data; for example, if the data to be written to the file are monthly, the year and month components of time are needed and the day and hour components are ignored.

Before the data can be written to the file, the user must also define a horizontal longitude-latitude grid by the use of the *grid('name','type')* command. Here, *name* is a unique name for the grid, and *type* can be one of the following: GAUSSIAN for non-linearly spaced Gaussian grids, LINEAR for evenly spaced grids, or GENERIC. To set the timestat, for the file the command *time('stat')* is used, where *stat* can be one of the following: AVERAGE, INSTANT, ACCUM, or OTHER.

Once this data setup is complete, the data can be written to the file with the command *file.latswrite(slab)*, where *file* is an opened LATS file object and *slab* is the slab of data that is to be written to the file. When the user has finished writing data, the file can be closed with the command *file.latsclose()*, where *file* is the LATS file object that is to be closed. Examples of the use of these commands are shown in Figure 8.

```

CDAT> year(1979,1988)

CDAT> month(1,12)

CDAT> day(0,0)

CDAT> hour(0,0)

CDAT> grid('mygrid','gaussian')

CDAT> vertical('pressure','plev')

CDAT> time('average')

CDAT> fl.latswrite(s)
The slab of data named s has be written to latsfile.nc

CDAT> fl.latsclose()
The lats file latsfile.nc is now closed

```

Figure 8. LATS Examples of the year, month, day, hour, grid, vertical, time, latswrite, and latsclose Functions

VCS Module

The VCS module was designed to allow the data retrieved from the cdunif module to be graphically displayed in CDAT. The module is a Python extension and was written in C. The module is the interface between VCS and Python.

The VCS module has several commands that give the user the flexibility to define how to display data. The `x=vcs.init()` command initializes the VCS module, where `x` becomes the VCS object. The initialization command must be executed before any other VCS commands can be implemented.

The `x.list('element')` command is used to list the data, and/or the VCS templates, graphics methods, and secondary elements. The VCS display is set

with the `x.set('type','name')` command, where *type* can be either 'template', 'graphics method', or 'secondary element' and *name* is the associated name of this type.

If the set command is not executed, VCS will use default attributes to display the data in the VCS canvas object. The VCS canvas object can be displayed either in landscape or portrait mode. The `x.page()` command is used to toggle between landscape and portrait display modes. Examples of the use of these VCS commands are shown in Figure 9.

```

CDAT> x=vcs.init()
'Template' is currently set to P_default.
Graphics method 'Boxfill' is currently set to Gfb_default

CDAT> x.list('template')
*****Template Names List*****
( 1): default          default_dud          por_topof3
( 4): por_topof3_dud    por_midof3
      por_midof3_dud
( 7): por_botof3        por_botof3_dud          AMIP
(10): AMIPDUD           AMIP_1of2              AMIP_2of2
(13): AMIPDUD_1of2      AMIPDUD_2of2           trop
(16): book_1of2         book_2of2              PMIP_2of4
(19): PMIP_3of4         PMIP_4of4              PMIP_1of4
*****End Template Names List*****

CDAT> x.set('template','AMIP')
Default 'Template' now set to P_AMIP

CDAT> x.page()

```

Figure 9. VCS Examples of the init, list, set, and page Functions

To display a slab of data on the VCS canvas object, the command `x.plot(slab)` is executed, where *slab* is the slab of data to be displayed. With no other arguments passed to the plot command, the slab is plotted using the

default graphics method and template. The following arguments can also be passed to the *plot* command: *template*, which defines where the data is to be displayed, *graphics type*, which defines how the data will be displayed, and *graphic name*, which defines the name of the graphics type. The user can also define which dimensions of the slab to display with the *x.grid(xbeg,xend,ybeg,yend,zbeg,zend)* command. The grid command executed in this manner will display the x dimension values from *xbeg* to *xend*, the y dimension values from *ybeg* to *yend*, and the z dimension values from *zbeg* to *zend*. To reset the grid to the values contained in the slab the *x.resetgrid()* is executed. To clear the VCS canvas object of any slab of data displayed on the canvas, the *x.clear()* command is used. To close the VCS canvas object, the command *x.close()* is executed. Figure 10 shows examples of the use of these commands.

Figure 10. VCS Examples of the plot, grid, resetgrid, clear, and close Functions

The CDAT user can create and run animations by executing the command *animationgui*, which will pop up the VCS animation GUI. The user can also change the color map for the VCS display with two different commands. Execution of *colormapgui* will pop up the VCS colormap GUI for changing colormaps. The command *x.colormap('name')* then is used to set the name of the desired colormap that is defined in the initial.attributes file of VCS. Examples of these commands are shown in Figure 11.

Figure 11. VCS Examples of the animationgui, colormapgui, colormap Functions

The VCS module also includes commands to create cgm and raster files. The `x.cgm('filename')` command creates cgm files that can be output through VCS to a printer. The `x.raster('filename')` command creates a raster file that can be used in an animation. For both of these commands, the default is to append data to the file each time the command is executed with the same filename. The user also can choose the option 'r' to replace the file instead of appending to the file. Through the VCS module, the CDAT user can also run scripts with the `x.script('scriptname')` command, where *scriptname* is the name of a script created in VCS. Examples of the use of these VCS commands are shown in Figure 12.

```
CDAT> x.cgm('/penguin0/susan/cdat/test')
Info - created CGM file (/penguin0/susan/cdat/test.cgm).
```

```

CDAT> x.raster('/penguin0/susan/cdat/test')
Info - create the raster Output file (/penguin0/susan/cdat/test.ras).

CDAT> x.script('clt.scr')

```

Figure 12. VCS Examples of the cgm, raster, and script Functions

PCMDI Module

The PCMDI module was designed to give the CDAT user the capabilities to run different functions used in climate applications. The module is a Python extension that was written in C. The module is the interface between Python and the routines developed by PCMDI's climate scientists.

Besides including climate application functions, the PCMDI module contains some general purpose functions. The command `cd('directory')` is used like the UNIX `cd` command to change the current directory. The `pwd()` command returns the present working directory analogous to the UNIX `pwd` command. The `ls()` command is used to list all of the files and directories in a given directory, as in the UNIX `ls` command. Figure 13 shows examples of the use of the general purpose commands.

```

CDAT> pwd()
/penguin1/cdat/cdati

CDAT> cd('../')
/penguin1/cdat

CDAT> ls()
***** SUB - DIRECTORIES *****
./
../
cdati/
Python-1.4/
***** END SUB - DIRECTORIES *****

```

```

***** FILES *****
oct1.ras*
latsfile.nc*
regrid.py*
***** END FILES *****

```

Figure 13. PCMDI Examples of the cd, pwd, and ls Functions

In addition, the *slabinfo(slab)* function lists all of the attributes and dimensions of the variable contained in a slab. An example of the use of the *slabinfo* function is shown in Figure 14.

The PCMDI module also includes two routines that are used to regrid a slab of data to a different grid. The function *maskin('slab', 'type')* is used if the slab to be regridded is also to be masked on either land or ocean values. The *slab* parameter denotes a slab of data the same size as the slab to be regridded. The *type* parameter specifies masking over 'land' or 'ocean'. The *newslab=regrid('slab1', 'grid1', 'slab2', 'grid2')* command is used to regrid *slab1* to the grid of *slab2*. The *grid1* and *grid2* parameters can

```

CDAT> pcmdi.slabinfo(csu)
*** Slab Information ***
Slab name: evs
Variable name: evs
File name: /penguin1/susan/jerry/evs/evs_csu.nc
Data type: Float
Number of dimensions: 3
longitude: degrees
length: 144
start: 0, end: 357.5
latitude: degrees
length: 73
start: -90, end: 90

```

```

season:
length: 1
start: 1, end: 1
Number of attributes: 7
source: CSU CSU91 4x5L17 AMIP 10 Year Simulation ( 1979-1988
)
name: evs
title: DJF10 Mean Surface Evaporation (All D,J and Fs)
units: mm/day
type:
date: 2/28/95
time: 10:56:03

```

Figure 14. PCMDI Example of the slabinfo Function

either be the model that created the data or one of the following options:

‘gaussian’, ‘linear’, ‘equalarea’, or ‘gislat’. Each of these grids defines a grid that the slab is stored on. The regrid function will return *newslab* which will include the data stored in *slab1*, but regridded to *grid2*. Figure 15 shows the *maskin* and *regrid* functions. The *shape* command is used in Figure 15 to show the dimension size of the slab before it goes through the regrid function.

```

CDAT> obsf=cu.open('/penguin1/susan/obs/tas.lw.ll.sc.av.54.ctl')
CDAT> csufl=cu.open('/penguin1/susan/jerry/evs/evs_csu.nc')
CDAT> obs=obsf.getslab('tas')
CDAT> obs.shape
(4, 46, 72)

CDAT> csu=csufl.getslab('evs')

CDAT> csu.shape
(1, 73, 144)

CDAT> csur=pcmdi.pregrid(csu,'linear',obs,'linear')
CDAT> csur.shape
(1, 46, 72)

```

Figure 15. PCMDI Examples of the maskin and regrid Functions

The CDAT user has the ability to write out slabs of data in either DRS, netCDF, or HDF formats. The command *writenetcdf(slab, 'filename', ['mode'])* creates a netCDF file. The command *writehdf(slab, 'filename', ['mode'])* creates a HDF file, and the command *writedrs(slab, 'filename', ['mode'])* creates a DRS file. All of these routines have the ability to either replace a file or append to an existing file. (The default mode is to append to a file; with the 'r' mode the file will be overwritten). Examples of the use of these routines to output slabs of data are shown in Figure 16.

```

CDAT> writenetcdf(s,'/penguin0/susan/cdat/testnetcdf')
Info - created the netCDF file (/penguin0/susan/cdat/testnetcdf).

CDAT> writehdf(s,'/penguin0/susan/cdat/testhdf', 'a')
Info - appended to HDF file (/penguin0/susan/cdat/testhdf).

CDAT> writedrs(s,'/penguin0/susan/cdat/testdrs')
Info - created the DRS file (/penguin0/susan/cdat/testdrs).

```

Figure 16. PCMDI Examples of the writenetcdf, writehdf, and writedrs Functions

CDAT Scripting

CDAT and Python scripts can be executed directly in CDAT, obviating the need to repeatedly enter commands at the CDAT prompt. To execute a CDAT or Python script, the user opens the script editor window, and opens the file that is to be executed. The user can make changes to the opened script file and save the changes to a different file, or overwrite the existing file. Like Python scripts, CDAT scripts end with the extension .py. Figure 17 shows an example of a CDAT script and Figure 18 shows an example of a Python script.

```
obsf=cu.open('/pcmdi/obs/mo/psl/rnl_ecm/sc/psl.rnl_ecm.sc.7901.8812.sfcctl')
derfl=cu.open('/pcmdi/amipwrk/staff/watts/amipstuff/der/DJF1079_88mm.dic')
obs=obsf.getslab('psl')
der=derfl.getslab('mrso')
derr=regrid(der,'der',obs,'linear')
x=vcs.init()
x.plot(derr,'AMIP', 'isofill', 'AMIP_mrso')
```

Figure 17. CDAT Script Example

The CDAT script in Figure 17 uses `cdunif` to open two different files, then retrieves a slab of data from the two opened `cdunif` file objects. Next, the two slabs of data are sent to the `regrid` function, the `der` slab is returned in the new slab `derr` on `obs` grid. Then VCS is initialized so the new slab of data can be displayed with the `plot` command. The parameters to the `plot` command are the slab of data, the display template `AMIP`, the graphics method `isofill`, and the `isofill` type `AMIP_mrso`.

The Python script in Figure 18 searches the current directory for all files ending in .py. The script then opens all of the files, and outputs the data to the output window. Each file's contents are separated by the defined marker.

```
import glob
marker = '::::::'
for name in glob.glob('*.py'):
    input = open(name, 'r')
    print marker + name
    print input.read(),
```

Figure 18. Python Script Example

CHAPTER IV

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

Summary

CDAT was developed to help climate scientists to analyze climate data. CDAT will enable scientists to be more productive with their research, as it will greatly reduce the effort of reading, writing, manipulating, and graphically displaying these data.

Conclusions

PCMDI's principal goal is to expedite the diagnosis and intercomparison of climate model data. The software system developed by PCMDI will greatly simplify this process. The system is designed to be simple and flexible enough to be readily expanded in the future. The PCMDI software system currently includes: CDAT, which manipulates data and provides scientists with a powerful suite of tools to analyze climate data; VCS, which is a graphics package used to display, animate, and manipulate data; CDMS, which is a database system that automatically locates and extracts data from files; and Python which is the object-oriented scripting language used to interact with

PCMDI software system. All of these packages can be run as standalone processes.

Recommendations

CDAT is currently being tested by PCMDI scientists; it will be released to outside scientists once PCMDI scientists have approved of the commands and syntax.

CDAT is currently available for use on SGI IRIX 5.2, 5.3 or 6.1 operating systems.

We will begin to port CDAT to the other platforms as the need arises.

New modules will continue to be added to CDAT, increasing the already powerful capabilities. When CDMS is released, a module will be developed to interface CDMS and Python. As more scientific routines are supplied, they will be added to the PCMDI or NCAR modules. A wrapper will also be included to allow the user to import customized Fortran and C functions into CDAT. A module will be developed to ingest, process, and quality control the data that is sent to PCMDI.

REFERENCES

REFERENCES

1. Guido van Rossum, "Introduction to Python for UNIX/C Programmers", Copyright 1993, pp. 1-8.
2. "Common Gateway Interface", World Wide Web, <http://hoohoo.ncsa.uiuc.edu/cgi/intro.html>.
3. Mark Lutz , "Programming Python", O'Reilly & Associates, Inc., Sebastopol, Calif,
4. Dean Williams, "Visualization and Computation System (VCS)", World Wide Web, <http://www-pcmdi.llnl.gov/williams/vcs/vcshome.html>.
5. "GrADS", World Wide Web Home Page, <http://grads.iges.org/grads/head.html>.
6. "NCAR GRAPHICS", World Wide Web Home Page, <http://ngwww.uvar.edu>.
7. Donald W. Denbo, "netCDF Calculator Users Guide (DRAFT) Version 2.0, Pacific Northwest Laboratories, 1994, pp. 1-1-1-2, 7-1-7-5.
8. "X Graphical Kernel System (XGKS)", University of Illinois, Urbana, Illinois, <http://www.tue.nl/urc/wa/graphics/xgks.html>
9. Alan Southerton, "The Shell Hackers Guide to X and Motif", Wiley Professional Computing, Copyright 1994.
10. Chuck Musciano & Bill Kennedy, "HTML: The Definitive Guide", O'Reilly & Associates, Inc., Sebastopol, Calif, 1997.
11. "netCDF", World Wide Web Home Page, <http://www.unidata.ucar.edu/packages/netcdf/>.

12. “HDF”, World Wide Web Home Page,
<http://www.ncsa.uiuc.edu/SDG/Software/HDF/HDFIntro.html>.

APPENDIX A

LIST OF ACRONYMS

Atmospheric Model Intercomparison Project	AMIP
Application Programmers Interface	API
Climate Data Analysis Tool	CDAT
Climate Data Management System	CDMS
Command Data UNiform InterFace	cdunif
Common Gateway Interface	CGI
Computer Graphics Metafile	CGM
Center for Ocean-Land-Atmosphere Studies	COLA
Cooperative Ocean-Atmosphere Research Data Service	COARDS
Corporation for National Research Initiatives	CNRI
Data Retrieval and Storage	DRS
Graphical User Interface	GUI
Grid Analysis and Display System	GrADS
GRIdded Binary	GRIB
Hierarchical Data Format	HDF
Hyper Text Markup Language	HTML
Hyper Text Transfer Protocol	HTTP
Lawrence Livermore National Laboratory	LLNL
Library of AMIP data Transmission Standards	LATS
National Center for Atmospheric Research	NCAR
Network Common Data Format	netCDF
Program for Climate Model Diagnosis and Intercomparison	PCMDI
Visualization and Computation System	VCS
World Meteorological Organization	WMO
X Graphical Kernel System	XGKS

APPENDIX B

LIST OF CDAT FUNCTIONS

cdunif commands:

```
f=cu.open('filename')
f.listdim()                dimlist = f.listdim()
f.listdim('variable')     dimlist = f.listdim('variable')
f.listvar()                varlist = f.listvar()
f.listatt('variable')      attlist = f.listatt('variable')
att=f.getatt('variable','attribute')
f.defvar('variable')
f.describe('variable')     or ('variable','all')
d1=cu.setdim(first_value,last_value)
dim=f.dimlist('dimension name')
dim=f.dimarray('dimension name')
s=f.getslab('variable')
s=f.getslab('variable', d1, d2)
s=f.getslab('variable',0,360,-90,90)
s=f.getslab('variable',0.,360.,-90.,90.)
s=f.getslab('variable','.',-90,90,':')
f.close()
```

LATS commands:

`fldescr('center','model','comments')`

`convention('type')`

`calendar('type')`

`frequency('type')`

`delta(increment)`

`fl=latscreate('filename')`

`year(first,last)`

`month(first,last)`

`day(first,last)`

`hour(first,last)`

`grid('grid name','grid type')`

`vertical('vertical name','vertical type')`

`time('type')`

`fl.latswrite(slab1,...,slabn)`

`fl.latsclose()`

VCS commands:

`x=vcs.init()`

`x.open()`

`x.grid(firsti1,last1,...,firstn,lastn)`

`x.resetgrid()`

`x.plot(slab, ['tmeplate'], ['type'], ['graphics'])`

`x.clear()`

`x.close()`

`x.page()`

`x.colormap('colormap name')`

`x.list(['data'] ['template'] ['graphics'])` or any secondary attributes

`x.set('template','graphics')`

`x.script('filename.scr')`

`x.cgm('filename','mode')` r=replace a=append no mode defaults to append

`x.raster('filename','mode')` r=replace a=append no mode defaults to append

`colormapgui`

`animationgui`

CDAT commands:

exit

quit

emacs('filename')

vi('filename')

PCMDI commands:

maskin(slab,'mask') mask = land or ocean

newslab=regrid(slab1,grid1,slab2,slab2)

slabinfo(slab)

pwd()

ls()

cd('directory')

writenetcdf(slab,['filename'],['mode']) mode = r replace or a append

writehdf(slab,['filename'],['mode']) mode = r replace or a append

writedrs(slab,['filename'],['mode']) mode = r replace or a append

CDAT math commands:

`+ - * /`

`dot(slab1,slab2)`

`sqrt(slab)`

`log(slab)`

`log10(slab)`

`int(slab)`

`float(slab)`

`long(slab)`

`sin(slab)`

`cos(slab)`

`tan(slab)`

`acos(slab)`

`asin(slab)`

`atan(slab)`

`cosh(slab)`

`sinh(slab)`

`tanh(slab)`

`acosh(slab)`

`asinh(slab)`

`atanh(slab)`

`slab.shape`

`newslab=slab[0,0,:,:]` takes 4d to 2d

`newslab=add.reduce(slab)`

`newslab=transpose(slab)`

APPENDIX C

EXAMPLE SCRIPTS

Script using cdunif module functions:

```
f=cu.open('clt.nc')
f.listdim()
f.listvar()
f.listatt('clt')
units=f.getatt('clt','units')
f.defvar('clt')
f.describe('clt')
d1=cu.setdim(-180,180)
d2=cu.setdim(-50,50)
dim1=f.dimlist('latitude')
dim2=f.dimarray('latitude')
s1=f.getslab('clt')
s2=f.getslab('clt', d1, d2)
s3=f.getslab('clt',0,360,-90,90)
s4=f.getslab('clt','.',-50,50,3,3)
f.close()
```

Script using lats module functions:

```
f=cuopen(clt.nc)
s=f.getslab(clt,-180,175,-90,85)
fldescr(BMRC,BMR2,)
convention(pcmdi)
calendar(standard)
frequency(monthly)
delta(1)
fl=latscreate(latsfile)
year(1979,1988)
month(1,12)
day(0,0)
hour(0,0)
grid(mygrid,gaussian)
vertical(pressure,plev)
time(average)
fl.latswrite(s)
fl.latsclose
```

Script using vcs module functions:

```
f=cu.open('clt.nc')
s=f.getslab('clt')
x=vcs.init()
x.list('template')
x.plot(s)
x.clear()
x.grid(0,10,5,10)
x.plot(s)
x.clear()
x.resetgrid()
x.colormap('AMIP')
x.set('isofill','AMIP_clt')
x.plot(s)
x.clear()
x.page()
x.cgm('demo')
x.plot(s)
x.clear()
```

Script using pcmdi module functions:

```
pwd()
ls()
cd('../')

obsf=cu.open('/pcmdi/obs/mo/psl/rnl_ecm/sc/psl.rnl_ecm.sc.7901.8812.sfc.ct
l')
derfl=cu.open('/pcmdi/amipwrk/staff/watts/amipstuff/der/DJF1079_88mm.
dic')
obs=obsf.getslab('psl')
obs.shape
der=derfl.getslab('evs')
der.shape
derr=regrid(der,'der',obs,'linear')
derr.shape
x=vcs.init()
x.plot(derr)
```

Script using numeric module functions:

```
f=cu.open('clt.nc')
s=f.getslab('clt')
s.shape
t=add.reduce(s)
t.shape
```

APPENDIX D

CDAT GUI

CDAT INPUT WINDOW

CDAT OUTPUT WINDOW

CDAT GRAPHICS CANVAS

CDAT SCRIPT EDITOR WINDOW

CDAT HELP WINDOW

CDAT ANIMATION WINDOW

CDAT COLORMAP EDITOR WINDOW

Technical Information Department • Lawrence Livermore National Laboratory
University of California • Livermore, California 94551