Documentation for the Three-Dimensional Parameterized Chemistry and Transport Model

J. Eric Nielsen

Emergent Information Technologies, Incorporated
Atmospheric Chemistry and Dynamics Branch
NASA Goddard Space Flight Center
Greenbelt, Maryland, USA

November 29, 2000

Contents

1 Introduction **4**

2 Finding the code **5**

3 Managing the integration **5**
3.1 Operations script 6
3.2 Finding the dynamics 13
3.3 Checking the vailability of Unitree and helios1 18

4 External files **19**
4.1 Job log file 19
4.2 Constituent history files 21
4.2.1 Sample IDL reader 21
4.2.2 Content 22
4.3 Mass flux divergence history files 25
4.4 Source dynamics 25
4.5 Cumulus transport parameterization 26
4.6 Startup and Restart files 26
4.7 Parameterized chemistry 28

5 Compilation **28**

6 Setting the Parameters **29**
6.1 Operations script 29
6.2 modules.f 32

7 Adding or Deleting Constituents **32**
7.1 Operations script 33
7.2 modules.f 33

8 Running the model **33**

9 Altering the source code **35**

10 Programming notes **35**
10.1 Test modes 36
10.2 Vertical discretization 36
10.3 Mass conservation 36
10.4 Arrays AP and BP for cldmx and tpcore7 37
10.5 Memory faults in the SGIs 37

11 References **38**

12 Appendix **39**
12.1 Variable list for modules.f 39
12.2 Main program variable list 41
12.3 List of subroutines 44
12.4 Obtaining copies of this document 46

1 Introduction

The three-dimensional (3D) parameterized chemistry and transport model (PCTM) solves the constituent continuity equation

$$\frac{\partial \chi}{\partial t} = -\mathbf{V} \cdot \nabla \chi + P - \mathcal{L}(\chi) \tag{1}$$

where χ is the constituent mixing ratio, t is time, $\mathbf{V}$ is the total velocity vector, and P and $\mathcal{L}$ are the production and loss rates, respectively [*Douglass et al.*, 1989, *Rood et al.*, 1991]. Solution of Equation 1 is accomplished by *process splitting* [*Rood*, 1987; *McRea et al.*, 1982]. First, advection produces an intermediate constituent configuration, χ^*. Equation 1 is then written as

$$\frac{\chi(t+\Delta t) - \chi(t)}{\Delta t} = \frac{\chi^* - \chi(t)}{\Delta t} + P - \mathcal{L}\chi(t+\Delta t) \tag{2}$$

and the photochemical parameterization is incorporated by solving

$$\chi(t+\Delta t) = (\chi^* + P\Delta t) \,/\, (1 + L\Delta t)\,. \tag{3}$$

L is the loss frequency and arises from a Taylor's Series expansion of the loss term.

The advection scheme is flux-form semi-Lagrangian and is described by *Lin and Rood* [1996]. The dynamics which drive the transport are acquired from an external source, usually from archived wind, temperature, and surface pressure data sets. The production and loss rates for each active constituent are also generally taken from an external source. In addition to carrying photochemically active constituents, the PCTM is able to transport inert compounds in the same experiment. The PCTM is flexible, allowing each user to incorporate the source dynamics of their choice, employ various production and loss rates, and customize the initial conditions by replacing only the respective default routines. The other modules, hopefully the majority, are accessed by reference in a *makefile*, so the user need not be responsible for their maintenance.

Most of this manual is concerned with the operational aspects of the PCTM. These include execution of the model, description of the external datasets, compilation, adding or deleting constituents, and references for variables and subroutines. The user of the model is encouraged to review each section, and should not expect to successfully integrate the model without a comprehensive understanding of both the source code itself and the operations requirements. There are, for example, potential restrictions on the length of each experiment and the number of allowed constituents based on the availability of logical unit numbers. On the other hand, those interested only in analyzing the PCTM's generated datasets may skip directly to Section 4.2, where the constituent history files are described and the location of a simple procedure for reading the data is given.

2 Finding the code

Code for the default version of the PCTM, including fortran source routines and operations scripts, is found in

/home/nielsen/ctm/ffsl/pchem/default,

where ffsl is an acronym for "Flux-form semi-Lagrangian." A few of the lower-level routines common to other uses, such as mathematical library routines, are not located in this directory. The complete listing is obtained by examining the *makefile* which is used to compile the code and generate the executable:

/home/nielsen/ctm/ffsl/pchem/default/makefile_default.

The *makefile* is described in Section 5, where instructions on compilation are detailed. That section is not intended to offer an extensive treatment of the use of makefiles. But it can serve as a primer, and those not familiar with makefiles will soon become acquainted with their utility.

The script which executes the default version of the model is located at

/home/nielsen/ctm/ffsl/pchem/default/run.default.

It is intended to do as much of the work to execute the model as possible without intervention. Included are the tasks of automatic file name generation and assignment, acquisition of source dynamics datasets, and archival of the generated history files. Before operating the PCTM, the user must perform a few tasks in preparation, like setting the run-time parameters, and must posess the necessary accounts on both the local workstation cluster and on Unitree. But with these procedural requirements satisfied, and after careful consideration of the run-time parameters, the script will run independently. The operation of the model is extensively described in the next section.

Procedures for reading the PCTM's history files with IDL (Interactive Data Language, ©Research Systems, Inc.) can be found at:

/home/nielsen/tools/idl/pctm_sample_read.pro

and

/home/nielsen/tools/idl/pctmr3d.pro.

The former is a simple driver routine which demonstrates how to specify the few parameters which control the reader, *pctmr3d.pro*. That is, *pctmr3d* is the procedure around which to build post-processing capabilities. Section 4.2 details the structure of the history files and explains more about these two routines.

3 Managing the integration

In order to minimize user interaction with the model, provide for the orderly arrangement of input and output files, and enable repetitive submission of jobs, careful study of the script at

/home/nielsen/ctm/ffsl/pchem/default/run.default

is encouraged. This section highlights the capabilities and limitations of the operations procedure incorporated into this script. In particular, the peculiarities for recognizing the source winds datasets and the naming conventions for the history and restart files are discussed.

Before beginning, the user must designate a working directory. This directory is where the PCTM's operations and file management take place. Because of storage requirements, this directory should reside on a data disk. For example,

/misc/abc0n/username.

The user should not use his or her home directory for executing the PCTM. In the execution script, the working directory is given the variable name `$workdir`.

The default PCTM also assumes that the user has an account on Unitree, Goddard's mass storage system. This is where we archive the PCTM's history files. A Unitree file requirement must be met in order to test its availablility when data transfer is requested. See Section 3.3, where the routine which checks the status of Unitree, *phonedirac*, is discussed.

3.1 Operations script

The first several lines of the script are reserved for run-time parameters and the comments which describe them. These are detailed in Section 6.1. Here, we instead concentrate on the functionality of the script. Each of the run-time parameters is declared as a script variable, and in addition to their roles in the execution of the script, the run-time parameters are also read by the PCTM source code.

The user-defined script variable `$testmode` deserves mention. Test modes are activated when `testmode > 0`. Otherwise production mode is assumed. In test mode, the user may do some rudimentary debugging without invoking the file maintenance (scrubbing) characteristics of the production mode. Hence, wind source files, history files, and startup files, which are removed at the end of each production integration, are kept when testing is invoked.

After the script digests the user-supplied definitions, it immediately changes to the working directory

```
cd $workdir.
```

Next, we right-justify the experiment number, `$exp`, by expanding to three digits and zero-filling:

```
exp=$nexp0
if [ $nexp0 -lt 100 ] ; then  exp=0‘echo $nexp0 | cut -c1-2‘ ; fi
if [ $nexp0 -lt  10 ] ; then exp=00‘echo $nexp0 | cut -c1‘   ; fi
```

Immediately we complete the path names of the restart and archive directories on Unitree by appending the experiment number to the script variables supplied by the user.

The main while-do loop, which is executed for `$numruns` iterations now begins execution:

```
while [ $iter8n -le $numruns ] ; do.
```

With each pass through this loop, a distinct and sequential PCTM experiment of length `$runlength` days is completed. Thus, the primary tasks of the while-do loop are keeping track of time, finding the winds for the time interval of the current sequential run, and naming, assigning, and managing datasets. The script assumes that storage for the source winds datasets and for the generated history and restart files is available on Unitree. To allow for file assignment, the following two commands are required

```
FILENV=.assign
export FILENV.
```

These create the file *.assign* on the working directory. This file survives for the life of the current sequential integration, but is removed before the next pass through the loop.

The time interval for the current integration is easily found by adding `$runlength` to the beginning year, month and day. This is done by *caladvance* through an algorithm which simply steps through the calendar day-by-day:

```
$execdir/caladvance $runlength $byear $bmonth $bday 1 .
```

It keeps track of day number, month number, and year number. Accounting is done for leap years and the result is good for the years 1901 through 2099. After *caladvance* determines the ending date and time, it writes the result to a file named *newdate* and the ending date is subsequently awked into the script:

```
eyear=`cat newdate | awk  '{ print $1 }'`
emonth=`cat newdate | awk  '{ print $2 }'`
eday=`cat newdate | awk  '{ print $3 }'`.
```

The restart flag `rstar` is then set by comparing the beginning date and time to the date and time of the initial condition. If the two time stamps are identical, then an initial condition is assumed and the flag is set to `.false.`:

```
if [ $t1 -eq $t2 ] ; then
 rstar=.false. rflag=0
else
 rstar=.true. rflag=1
fi.
```

The time-stamp variables are then used to locate the source dynamics datasets which blanket the time interval for the current sequential integration. These include the winds for transport and, when the switch `cloud = .true.`, the cloud mass flux and detrainment. These datasets are located by the *selector* and the *windfinder*, which are described in Section 3.2. The selector provides a return code which is tested to indicate success in finding the datasets,

```
$dir4wind/selector_strat $byear $bmonth $bday $cloud $dir4wind 1 $runlength
if [ $? -ne 0 ] ; then
 echo FATAL ERROR detected in SELECTOR
 exit 1
fi.
```

It also returns three important variables for this script. They are the number of each class of dynamics files which will be opened during this sequential run, `$numwinds`, and the date and time of the first wind "observation" on the first of the source datasets, `$yymmdd0` and `$hh0`. These are also awked into the script:

```
numwinds=‘cat numwindfiles | awk  '{ print $1 }'‘
yymmdd0=‘cat icdate | awk '{ print $1 }'‘
hh0=‘cat icdate | awk '{ print $2 }'‘
```

To speed the integrations, the next section of the code "preselects" the dynamics files needed for the next sequential submission. This is accomplished by advancing the clock by `$runlength` days to determine the starting date. Then the selector is called as before, but with the logical unit assignment switch turned off. The job is submitted to background with the `at` utility, and begins running while the current integration is in progress. The selector creates the file *selector.status* when it becomes active and erases it when it finishes. Immediately following the end of the PCTM's execution, we check for the existence of the the status file and pause exectution of the script until it disappears. Preselection is invoked only in production mode. The selector and wind finder are described in Section 3.2.

Next, the script defines file names and assigns the files to logical unit numbers to the "non-dynamics" datasets. A list of files and their logical unit numbers is given in Table 1. The file names conform to the naming convention given by examples in that table. For example, the startup and restart file names look like

p034.restart.19980412z12.

The file names require two-digits for the month, day, and hour, and four digits for the year. Only the representation of the hours is incomplete at this point, so we expand and zero-fill:

```
if [ $bhh -lt 10 ] ; then bhh=0‘echo $bhh | cut -c1‘ ; fi
if [ $ehh -lt 10 ] ; then ehh=0‘echo $ehh | cut -c1‘ ; fi
```

Now we can find the file names for the startup and restart files, the run-time parameters file, and for standard output. Using `p` to indicate a "parameterized chemistry" experiment, we then derive permanent names for these files:

```
startup=p$exp.restart.$byymmdd'z'$bhh
restart=p$exp.restart.$eyymmdd'z'$ehh
logfile=p$exp.logfile.b$byymmdd'z'$bhh.e$eyymmdd'z'$ehh
runtimefile=p$exp.runtime.b$byymmdd'z'$bhh.
```

With the timekeeping functions completed and the wind source files found, all of the run-time parameters are now known and are concatenated into the `$runtimefile` file on the working directory. This dataset is read from logical unit number 9 by `subroutine setparams`:

```
cat > $runtimefile << EOF
⋮
EOF
```

The run time parameters file is completed by the commands which immediately follow. The history file write flag for each respective constituent and the name of the constituent are echoed into temporary files. These are concatenated to `$runtimefile`.

The script then attempts to find a startup file. This file contains a time stamp and the "current" distributions of the modeled constituents. If file `$startup` is not resident on the working directory then we assume that it can be transfered from the directory `$restartdir` on Unitree. Hence the test

```
if [ ! -r $startup ] ; then
⋮
fi
```

In general, startup and restart files have the same file structure. Thus a restart file, which is written at the end of execution, will serve as the startup file on the next run in the sequence. The only exception occurs when, for example, an IDL procedure is used to generate the *initial condition*. Please see the discussion on file assignment below and read Section 4.6 to learn more about startup and restart files.

We now assign logical unit numbers to the startup, restart, run time parameters, and production and loss rates files. The logical unit numbers are listed in Table 1 and the options for *assign* can be reviewed on the man page. The typical assign looks like

```
assign -a $filename -F $fspec fort.nn.
```

The script assumes that the first startup file, the initial condition, is generated by an IDL procedure, so the file format is `null`. The user will need to modify the script if this is not the case for their particular experiment. The model-generated restart files are of format `f77`. The production and loss rates files are assigned only if the `chemistry` switch is set to `.true.`. The list of production and loss rates files will vary from experiment to experiment.

Permanent filenames are now found for the constituent, temperature, vertical velocity, and potential vorticity history files. If the `fluxdiv` switch is activated, then names for the history files of the components of the mass flux divergence for each constituent are also defined. The constituent history file names look like

p034.n2o.b19940412z06.e19940511z00.

The history file names for the components of the mass flux divergence look like

p034.n2omfd.b19940412z06.e19940511z00.

The content of these files is described in Sections 4.2 and 4.3. The time of the first writes to the history files depends on `$rstar`. If `rstar=.false.`, the initial condition is written to the history files when the model begins exectution. If `rstar=.true.`, then the first write will not be executed until `$dthist` hours have elapsed from the restart time. For the mass flux divergence history files, the first write does not occur until `$dthist` hours have elapsed from the restart time. In both of these cases, the first set of records on each history file will have a time stamp which is not identical to the restart date and time. To retain the naming convention, we must invoke *caladvance* to find the correct date and time of the first write,

```
if [ $rflag -eq 1 ] ; then
⋮
 $clockdir/caladvance 1 $byear $bmonth $bday 1
 cyear=‘cat newdate | awk  ’{ print $1 }’‘
⋮
fi
```

The results are awked into the script.

With the date stamp known, we can now assemble the permanent history file names and assign logical unit numbers. In addition we now prepare for archiving by filling temporary files with file transfer *put* commands to be used after the PCTM terminates execution. These tasks are accomplished in a while-do loop which is executed `$numcon` times

```
while [ $count -le $numcon ] ; do
⋮
done.
```

Names are derived as above with the restart and log files, and the logical unit numbers are assigned by

```
filenam=p$exp.$content.b$cyymmdd’z’$chh.e$eyymmdd’z’$ehh
assign -a $workdir/$history -F null fort.$filenum.
```

The ftp *put*s, are echoed into temporary files. For example,

```
echo put $history > tmp.xn.
```

If the switch to store the components of the mass flux divergences is activated, then the above is repeated and files for the mass flux divergences are also assigned. The logical unit numbers are assigned consecutively and the association with the named constituents is as they are listed at the top of the script. As of this writing, the number of constituents, `$numcon`, must be ≤ 9. Logical unit numbers for constituents run from 10 through 18 and for the components of the mass flux divergence from 20 through 28. The put commands are concatenated into a single temporary file, `tmp.x1`.

The name and file assignments are next completed for the temperature if the switch `$storetk` is activated, for vertical wind (as computed by the transport subroutine *tpcore7m.f*) if the switch `$storew` is activated, and for the potential vorticity if the switch `$storepv` is activated. These history files are assigned unit numbers 69, 68 and 67, respectively. The file format `null` is chosen for all history files for ease of post-processing.

The PCTM can now be executed. The executable is created (See Section 5) as separate, prerequisite step and is expected to be located at `$workdir/$executable`. While in production mode, standard output is sent to `$logfile`

```
$executable > $logfile.
```

In test mode, however, the output is scrolled to the screen. Please see Section 8 for the differences between submitting the job in production and test modes.

Following execution, a test is performed to detect the existence of a core dump. If it is detected, then execution is immediately stopped. Next, a test determines whether the selector status file still exists. This file is "active" as long as the selector scipt is executing. In that case, a wait state is invoked for five minutes, and then the test is run again. When the selector status file diappears, the script procedes to its next tasks.

After normal end of execution we attempt to archive the history, restart, and log files. This involves several steps. First, we assemble a list of archive commands. The *put*s for the restart file and the logfile are echoed into temporary files

```
echo put $restart > tmp.x7
echo put $logfile > tmp.x8.
```

The several `tmp.x*` files are now concatenated into `tmp.y`. The latter, which is a list of put commands for the files to be achived, is subjected to the editor, and all occurrances of the string *put* are changed to *chmod 644*

```
cat << EOedits | ed -s tmp.y
1,$ s/put/chmod 644/
w
q
EOedits.
```

After editing `tmp.y` is moved to `tmp.x9`, the directory selection and *quit* commands are echoed into temporary files

```
echo cd $archivedir > tmp.x0
echo quit > tmp.xa,
```

and we are ready to assemble the file transfer command file by concatenating the temporary files in order:

```
cat tmp.x* > ftp.com.file.
```

A typical example, for which the temperature archive option is activated, might look like

```
cd /u3/y3jen/ctm/exp034
put p034.n2o.b19951015z06.e19951029z00
put p034.ch4.b19951015z06.e19951029z00
put p034.T.b19951015z06.e19951029z00
put p034.restart.19951029z00
put p034.logfile.b19951015z00.e19951029z00
chmod 644 p034.n2o.b19951015z06.e19951029z00
chmod 644 p034.ch4.b19951015z06.e19951029z00
chmod 644 p034.T.b19951015z06.e19951029z00
chmod 644 p034.restart.19951029z00
chmod 644 p034.logfile.b19951015z00.e19951029z00
quit.
```

Thus, after the files arrive at Unitree, their permission sets are changed to allow read access to your colleagues.

The minimum requirement for transfer of files is that Unitree is alive and well. This is determined by *phonedirac*, which is described in Section 3.3. If Unitree answers the phone, then *phonedirac*'s return code is set to zero and the script variable `$scrub` kept at its default configuration, which is 'on'. If Unitree does not respond, `$scrub` is changed to 'off', and no attempt is made to archive the history files.

```
scrub=1
$execdir/phonedirac
if [ $? -eq 1 ] ; then scrub=0 ; fi
if [ $scrub -eq 0 ] ; then
 echo WARNING: Unitree unresponsive.  Keeping history files.
else
```

In the case where `$scrub` is activated, the test mode indicator `$testmode` determines the fate of the history files. In production mode, `testmode=0`, the files are removed after archiving. At test level 1, `testmode=1`, the files are archived but not removed. At test level 2, `testmode=2`, the files are neither archived nor removed. The abbreviated decision tree is shown below with the file transfer commands piped to ftp

```
if [ $testmode -le 1 ] ; then
 echo Archiving history files to $archivedir ...
```

```
 cat ftp.com.file2 | ftp dirac 1021.
fi
⋮
 if [ ${saveflag[$count]} = '.true.' ] ; then
  history=p$exp.$content.b$cyymmdd'z'$chh.e$eyymmdd'z'$ehh
  echo $workdir/$history
  if [ $testmode -eq 0 ] ; then /bin/rm $workdir/$history ; fi
 fi
⋮
fi.
```

For the constituent and mass flux divergence files, this structure is wrapped in a while-do loop. Upon reaching the done statement, the structure is replicated for the optional history files. Afterward, the startup file is removed if production mode is enabled. When either of the testing modes are activated, the list of files to be removed is echoed to the screen.

In production mode, the source winds datasets which are not needed for the next sequential submission are removed. In either test mode, they are not removed. In the default configuration the integrations are ten days in length and the source winds datasets have one day of data on each. Hence the first ten files of the eleven needed for the current integration may be removed. In a while-do loop, the files are listed (in chronological order) and concatenated into the file `$filelist`. With each pass through the loop, the file listed at the head of `$filelist` is removed and an index is incremented. After ten passes, the ten files are gone:

```
index=1
while [ $index -le $limit ] ; do
 ls -1 stratf.prog.sig.t* > filelist
 filename=`head -1 filelist`
 echo $workdir/$filename
 /bin/rm $workdir/$filename
 index=`expr $index + 1`
done
```

In either test mode, the script simply prints the contents of `$filelist` for debugging.

The final task for the current iteration of the script is incrementing the iteration number and advancing the beginning date and time in anticipation of another sequential submission.

3.2 Finding the dynamics

The script(s) to locate and retrieve the driver dynamics for the default version are written for STRAT assimilated data. Other versions which access TRMM assimilation and FVCCM winds may be found in other `/pchem` subdirectories. Finding the STRAT dynamics begins with the *selector*,

/home/nielsen/ctm/ffsl/pchem/default/selector_strat.

The selector calls the *windfinder*,

/home/nielsen/ctm/ffsl/pchem/default/windfinder_strat,

once for retrieving the winds, temperatures, and surface pressures, and once for retrieving the cloud mass flux and detrainment. In the case of STRAT, the seclection is made between *dynamics* and *diagnostics*, respectively. A test is included to detect a failure, and execution will stop. For example:

```
$dir4wind/windfinder_strat $year $month $day $ok2assign $runlength dynamics
if [ $? -ne 0 ] ; then
 echo FATAL ERROR Unable to acquire dynamics from archive.
 if [ $? -eq 1 ] ; then echo Archive unaccessible ; fi
 if [ $? -eq 2 ] ; then echo Unable to acquire complete dataset ; fi
 echo ABNORMAL STOP
 /bin/rm selector.status
 exit 1
fi.
```

The wind finder then determines by date which datasets will be needed to successfully integrate the model through the default integration period. The wind finder is at

/home/nielsen/ctm/ffsl/pchem/default/windfinder_strat.

The datasets contain dynamics from a "final analysis" integration of the 46-level STRAT assimilated data model for 1996 through 1998. Dynamics source files are discussed in Section 4.4. Here, we concentrate on deriving the dataset names, obtaining the files, and assigning them to logical units.

The call list contains six variables: the year, day, and month, a flag which when set to one tells the script to assign the datasets to logical unit numbers after they are found, the length of the integration in days, and the data type to be acquired. The STRAT final analysis winds datasets contain one day's data per file, with data written at six-hour intervals. Hence the number of files of each data type which must be retrieved are

```
files2get=‘expr $runlength + 1‘.
```

The time stamp of the first set of records is 0:00 UTC. Thus the first required file will have a date stamp which is identical to that of the initial/restart condition. The date is found:

```
dsyear=‘echo $year | cut -c3-4‘ dsmonth=$month dsday=$day
date=$dsyear$dsmonth$dsday,
```

and the first logical unit number is declared depending on the value of `$datatype`

```
if [ $datatype = 'dynamics' ] ; then
 lun=70
⋮
elif [ $datatype = 'diagnostics' ] ; then
 lun=50
⋮
fi.
```

Next we enter the while-do loop which will attempt to find, verify the size of, and assign each required dataset. STRAT winds are stored on Unitree through 11 Dec 1997, and afterward are stored on *helios1* at NASA Ames. We test for the date, and assign the machine identification for the file transfers

```
machineid=dirac remotehost='dirac 1021'
if [ $date -ge 971200 ] ; then
 machineid=helios1
 remotehost=helios1.nas.nasa.gov
fi.
```

Similarly, the directories and the file sizes different. For the winds, temperatures, and surface pressures we need the *prog* files, indicated by the `dynamics` datatype

```
 if [ $datatype = 'dynamics' ] ; then
... geosdir=/u2/w3djl/stratf/prog dsgen=stratf.prog.sig.t
 if [ $date -ge 971200 ] ; then geosdir=/chuck/u2/lamich/stratf/prog ; fi.
```

When the `diagnostics` datatype is requested, a separate set of assignments are made for cloud flux and detrainment. The wind finder tests for file size, so the expected file sizes are given a declared variable, `$reqsize`. It is different for not only the `dynamics` and `diagnostics` data types, but also after dates in which the DAO made changes to the files' contents

```
 if [ $datatype = 'dynamics' ] ; then
  reqsize=69431568 ...
⋮
  if [ $date -ge 970806 ] ; then reqsize=69851440 ; fi
⋮
 elif [ $datatype = 'diagnostics' ] ; then
⋮
  reqsize=217307536 ...
⋮
  if [ $date -ge 970213 ] ; then \ldots reqsize=236604400 ; fi
⋮
 fi
```

Note also that for several dates the datasets are given specific sizes. These files represent data that is missing from the STRAT archive and have been recreated from the Code 916 science system in the case of the `dynamics` but remains missing for the `diagnostics`. These reconstructed files are located on */misc/ctm02/nielsen*, and the user must make his/her own copies on their `$workdir$` before integrating through the indicated dates.

The wind finder will look for a local copy of the desired dataset on `$sourcedir`, and if it does not exist then a file transfer from `$machineid` will be attempted. The remote and local file names are assembled after the above tests are completed. The remote file name is

```
$geosdir/$subdir/$file
```

and the local file name is

```
$sourcedir/$file.
```

If the local file does not exist then the script assumes that a copy is archived on Unitree if the date is before 12 Dec 1997 and on *helios1* afterward. The current status and availability of the remote systems are tested by *phonedirac* and *phonehelios*, respectively. These scripts are described in the next section. If the remote machine is up and running, the script will notify the user and attempt to transfer the missing file

```
if [ ! -r $sourcedir/$file ] ; then
 echo Attempting ftp for $file
⋮
 $execdir/phone$machineid
 if [ $? -eq 1 ] ; then
  stamp=`date`
  echo Windfinder ends abnormally at $stamp
  retcode=1
  exit $retcode
 fi
ftp $remotehost << EOF
get $geosdir/$subdir/$file $sourcedir/$file
EOF
fi
```

The local copy of the dataset is tested for size. The actual file size is awked into a script variable and is compared to `$reqsize`:

```
filesize=`ls -l $sourcedir/$file | awk '{ print $5 }'`
if [ ${filesize:=0} -eq $reqsize ] ; then
```

If the expected value and the actual value are equal, then control is passed to the next section of the script. If not, or if the file does not exist and has no length designation to awk, the file is removed and another attempt to transfer the file is executed. If the file sizes do not match after `$maxtries` attempts, then the script exits with a return code of 2 after it issues a warning:

```
echo File: $sourcedir/$file has $filesize bytes.
echo Expected size is $reqsize bytes.  Retry.
/bin/rm $sourcedir/$file
if [ $attemptno -ge $maxtries ] ; then
 echo WARNING: Unable to acquire $file
 echo EXITING after $attemptno attempts.
 retcode=2
 exit $retcode.
```

A successful test of file size is followed by assignment of that file to a logical unit number if the flag `$ok2assign` is one:

```
if [ $ok2assign -eq 1 ] ; then
 echo Assigning $file to fort.$lun ...
 assign -a $file -F f77 fort.$lun.
fi.
```

The logical unit numbers, nn, are limited to the range

$$70 \leq nn \leq 84$$

for the winds and

$$50 \leq nn \leq 64$$

for the cloud mass flux and detrainment.

WARNING: This restriction imposes a limitation to the length of individual integrations of a sequence.

At the base of the while-do loop, the date is incremented by one day in preparation for retrieving the next file in the sequence

```
 $clockdir/caladvance 1 $year $dsmonth $dsday 1
 year=‘cat newdate | awk  ’{ print $1 }’ | cut -c1-4‘
 dsyear=‘cat newdate | awk  ’{ print $1 }’ | cut -c3-4‘
 dsmonth=‘cat newdate | awk  ’{ print $2 }’‘
 dsday=‘cat newdate | awk  ’{ print $3 }’‘
⋮
 date=$dsyear$dsmonth$dsday.
```

After `$files2get` iterations, copies of all files of the current `datatype` are available locally on `$sourcedir`.

Before finishing, and only when the logical unit number assignment switch is on, the script writes the date and time of the first "observation" from the first required dataset to the file *icdate*, and the total number of wind files needed to complete the current sequential submittal to the file *numwindfiles*. The PCTM uses these parameters to bypass the dynamics which precede the integration's start time, when necessary. The wind finder status file is removed immediately before exiting. The model reads the data in routines called by `subroutine readwinds`.

3.3 Checking the vailability of Unitree and helios1

Many experiments consist of several sequential submissions which integrate collectively for days or even weeks. Local disk space must be agressively managed during these integrations. In fact, we assume that there is only enough space for the log files and the currently required wind source, history, and restart datasets. Archival is a necessity, and Unitree represents the most viable alternative. Yet the avaiability of *dirac* for transfer of datasets at the beginning and the end of sequential submissions represents a important failure mode. *phonedirac* addresses the availability issue. It determines whether Unitree can transfer files. If not, then execution is paused and another determination is attempted after a specified time interval.

The user interacts with the script in two ways. First, the number of access attempts and the number of seconds which will lapse between them is declared. For example,

```
numtries=96
waittime=900
```

Second, and before using *phonedirac*, the user must create the file *access.number* on his/her home directory on Unitree. The content is one ascii numeral which is incrementable by this script. For example, 1. The script attempts to transfer this file from Unitree

```
ftp dirac 1021 << EOF1
get access.number
quit
EOF1.
```

Success is indicated by the local presence of the file. That is,

```
if [ -r access.number ] ; then.
```

The numeral is concatenated into a local script variable whose value is incremented. The local copy of *access.number* is erased and new copy is created which contains the incremented numeral. That is,

```
accno=‘cat access.number‘
accno=‘expr $accno + 1‘
/bin/rm access.number
echo $accno > access.number.
```

The new file is then transfered to Unitree, overwriting the original,

```
ftp dirac 1021 << EOF2
put access.number
quit
EOF2
```

effectively keeping a count of the number of times that *phonedirac* is invoked. Control is returned to the calling script with a '0' return code.

When Unitree is down, execution is paused

```
else
⋮
sleep $waittime.
```

After the prescribed number of seconds, the while-do loop counter is incremented and the process is repeated. If there is no response after `$numtries`, the user is notified and control is returned to the calling script with a '1' return code.

The procedure is the same for *helios1* and will not be repeated here. In the near future, however, only secure shell (ssh) interactions will be permitted with *helios1*. As of this writing, we have not implemented ssh in *phonehelios1*. But the user can use

/home/nielsen/ctm/ffsl/pchem/exec/phonegatun

as a template for ssh file transfers. Consult the system administrators for setting up ssh.

4 External files

Table 1 summarizes the external datasets required by the PCTM and the datasets which it creates. Included is a list of subroutine references for performing the input or output. Any files created prior to 13 May 1999 designate the year with only two digits. Each catagory of files is detailed in the following sections.

4.1 Job log file

The operations script assigns the name `$logfile` to the file into which standard output is directed. The name is based on the experiment number and the beginning and ending times for the current sequential integration. The log file is created as a result of the operations script's execution statement

```
pchem.default > $logfile.
```

For clarity, this file does not contain output from execution of the shell script commands (See Section 8). An example of the job log file can be found on Unitree at

/u3/y3jen/ctm/example/exp999/p999.logfile.b19971018z00.e19971020z00.

Table 1. External files listed by logical unit number				
Job log file				
Logical unit no.	Subroutine References	Script Variable name	Assign object	Contents
6	Numerous	`$logfile`	None	Standard output
Example: *p234.logfile.b19960203z00.e19960215z00*				
Constituent History files				
Logical unit no.	Subroutine References	Script Variable name	Assign object	Contents
10	writehistory	`$content`	*fort.10*	`$speciesname1`
11			*fort.11*	`$speciesname2`
⋮			⋮	⋮
18			*fort.18*	`$speciesname9`
67	writehistory	`epv`	*fort.68*	$-g(f+\mathbf{k}\cdot\nabla\times\mathbf{V})\frac{\partial\theta}{\partial p}$
68		`w`	*fort.68*	Vertical wind
69		`T`	*fort.69*	Temperature
Example: *p234.co2.b19960203z06.e19960215z00*				
Components of mass flux divergence history files				
Logical unit no.	Subroutine References	Script Variable name	Assign object	Contents
20	writefluxdiv	`$content$fluxtag`	*fort.20*	$\frac{\partial(\rho u)}{\partial x}$,$\frac{\partial(\rho v)}{\partial y}$,$\nabla\cdot(\rho\mathbf{V})$
21			*fort.21*	for respective specie,
⋮			⋮	cumulative
28			*fort.28*	
Example: *p234.co2mfd.b19960203z06.e19960215z00*				
Source dynamics				
Logical unit no.	Subroutine References	Script Variable name	Assign object	Contents
70	readwinds	`$sourcedir/$file`	*fort.70*	p_s,u,v,and T
71			*fort.71*	
⋮			⋮	
84			*fort.84*	
Example: *i.stratf_CTM.19970412.18*				
Cloud mass flux and detrainment				
Logical unit no.	Subroutine References	Script Variable name	Assign object	Contents
50	readcloud	`$fn`	*fort.50*	Cloud mass flux and
51			*fort.51*	detrainment
⋮			⋮	
64			*fort.64*	
Example: *stratf.psfcld.46.1a.b19960812z00.e19960818z18*				

Table 1 continued. External files listed by logical unit number				
Parameterized chemistry				
Logical unit no.	Subroutine References	Script Variable name	Assign object	Contents
85	ratecalc	None	*fort.85*	Production rates, P
86			*fort.86*	and/or loss rates, L
Example: */misc/chj02/nielsen/oxpl9129.hybln.dat*				
Startup and restart				
Logical unit no.	Subroutine References	Script Variable name	Assign object	Contents
95	readrestart startup	`$startup`	*fort.95*	Restart read Initial condition
95	writerestart	`$restart`	*fort.96*	Restart write
Example: *p234.restart.19960215z00*				

4.2 Constituent history files

The PCTM assigns each constituent its own history file, and examples of the naming convention for history files are given in Table 1. This section outlines the mechanics of writing the history files and the file's contents word-by-word. The detail is sufficient so that each user can design a routine to read a history file. Users with access to Interactive Data Language (IDL, ©Research Systems, Inc.) need not necessarily be concerned with the files' structure and instead may take advantage of the procedures outlined immediately below to read the files.

4.2.1 Sample IDL reader

The IDL procedure

/home/nielsen/tools/idl/pctmr3d.pro

may be used to read the PCTM's constituent, temperature, potential vorticity, vertical wind speed, and mass flux divergence history files. Placing the directory */home/nielsen/tools/idl* in your IDL path gives you access to this routine. All parameters and keywords are documented in the comments at the start of *pctmr3d.pro* and will not be described here. Since there are generally several "observations" or "time steps" on each file *pctmr3d.pro* controls opening and closing the history file. The user directs this action by assigning a value to the keyword `configure`. That is, before calling the reader, the user must first specify a file name and the value to `configure` (if the default is not acceptable). To clarify these requirements, a simple procedure may be copied from

/home/nielsen/tools/idl/pctm_sample_read.pro.

This routine places a call to *pctmr3d* in a loop which reads sequentially through a constituent history file. It illustrates how to set the `configure` keyword, provides for a sample printout of the run-time parameters, and plots both the surface pressure and the zonally averaged constituent mixing ratio at each time of "observation."

4.2.2 Content

During execution history files are assigned to logical unit numbers `nn` where $10 \leq nn \leq$ `$numcon`-1. The numbering is consecutive and the association with the named constituents is as they are listed by the user at the top of the operations script. The files contain 32-bit words and are assigned file format `-F null`. That is, there are no record mark delimiters.

Writes to the history files are done by `subroutine writehistory` at intervals which are specified by the user through the script variable `$dthist`. The information included at each write is, in order, a parameter summary, the surface pressure, and the constituent array itself. The parameter summary is called the "header," and is summarized in Tables 2a and 2b. All header assignments with the exception of word number 40 are done by `subroutine header`. The number of header words varies depending on the horizontal and vertical discretization and the number of constituents. Header word number 12 tells the user how many words are in the header. Specifically,

$$\texttt{lenhdr} = 40 + 2 \times \texttt{levels} + 1 + \texttt{lat} + 1 + \texttt{ncon}.$$

Trailing the header is the surface pressure which has dimensions (`long,lat+1`) and the respective constituent array, which has dimensions (`long,lat+1,levels`). Constituents are dimensionless since they are mixing ratios.

From the header, the user can easily determine the location of each data point. Vertically, the data lies at the layer means. Each layer mean vertical coordinate, σ, is read directly from the first `levels` words which follow word number forty. For the general case of a hybrid vertical discretization, the local pressure in the terrain-following portion of the domain is found by

$$p = p_i + \sigma \left(p_s - p_i \right), \tag{4}$$

where p is the pressure, p_s is the surface pressure, and p_i is the pressure at the interface between the terrain-following and isobaric parts of the vertical domain. There are `ktrop` layers in the terrain-following portion of the model. In the constant pressure portion of the model, the layers are bounded by isobaric surfaces and the local pressure is

$$p = p_i + \sigma \left(p_i - p_t \right), \tag{5}$$

where p_t is the model's lid pressure. Since the constant pressure domain has historically included the stratosphere, the number of isobaric levels is termed `kstrat` and is found by subtracting `ktrop` from `levels`. When the domain

Table 2a: First 40 words in the history header.		
Word Number	Code Label	Contents
1	`nexp`	Experiment number.
2		Spare -999.
3	`day`	Month, day, and day fraction (mmdd.ff) of local time of experiment execution.
4	`dayn0`	Month, day, and day fraction (mmdd.ff) of experiment beginning date and time.
5	`time/86400`	Model elapsed time (days).
6	`date1`	Month, day, and day fraction (mmdd.ff) of time stamp.
7	`dayn`	Calendar day, 0:00:00 UTC 1 Jan is day 1.00.
8	`ncon`	Number of constituents.
9	`long`	Number of longitudes.
10	`lat+1`	Number of latitudes.
11	`levels`	Number of layers.
12	`lenhdr`	Number of words in this header record.
13	`tdt`	Nonpolar dynamical time step, seconds.
14	`tdt`	Polar dynamical time step, seconds.
15	`cdt`	Time step length for chemistry, seconds.
16	`ktrop`	Number of sigma layers.
17	`pint`	σ-to-p interface pressure, p_i.
18	`ptop`	Top-layer pressure, p_t.
19	`firstlon`	Longitude of first point on latitude circle.
20	`histtd`	Vertical orientation of history files: 1=lid-to-ground, 0=ground-to-lid
21	`histint`	Time interval between history tape writes (sec). Before 22 Apr 1999, spare -999.
22		Spare -999..
23		Number of contituents whose history file write switch is set to `.true.`
24		Year (4-digit) of local time of experiment execution.
25		Year (4-digit) of experiment beginning date and time.
26		Year (4-digit) of time stamp.
27		Spare -999.
⋮	⋮	⋮
37		Spare -999.
38	`radius`	Radius of planet.
39	`omega`	Angular rate of rotation of planet.
40		Constituent number or 101=T, 102=dp/dt, 103=pv.

Table 2b: Remaining words in the history header.		
Word Numbers first/last	Code Label	Contents
41 40+`levels`	`sigma`	Layer mean σ.
41+`levels` 40+2`levels`+1	`sgint`	Layer interface σ.
41+2`levels`+1 40+2`levels`+1+`lat`+1	`latedge`$(180/\pi)$	Latitudes (degrees).
41+2`levels`+1+`lat`+1 40+2`levels`+1+`lat`+1+`ncon`		'1' when constituent is saved. Otherwise -999.

is terrain following everywhere, `ktrop`=`levels`, `kstrat`= 0, and $p_i = p_t$. For completeness, the next `levels+1` words list the vertical coordinate of the layer interfaces. *Tacaks, et al.*[1994] provides a complete description of the vertical cooridinate.

By applying relations above, the user can immediately determine whether the history file data is oriented "lid-to-ground" or "ground-to-lid." The operations script switch `histtd` allows the user to control the orientation *apriori*. This is potentially useful for accomodating existing sets of post-processing procedures. The value of `histtd` is reflected in binary form through word number 20 in the header. For a hybrid system, when `histtd` is 0(1), then then the σ to p interface is defined on the history files (only) by `sgint(ktrop+1)` (`sgint(kstrat+1)`).

Horizontally, the data mesh is regular in both directions. Header word number 19 gives the first longitude on each latitude circle. From there, the grid points are evenly spaced in longitude working toward the east. The latitudes are listed from south to north in the first `lat+1` words following the layer interface σs. Note that they are also evenly spaced. Word number forty is the constituent number for the file being currently read. The number corresponds exactly with the named constituents is as they are listed at the top of the execution script. For completeness, the final `ncon` words of the header review the state of the history file write switch for each constituent. If the switch is set to `.true.`, then the respective header word is set to a floating-point '1'.

History files for the vertical wind, temperature, and potential vorticity are written when their respective switches are activated (See Section 6.1.). The file structure is identical to that of the constituent files. Vertical wind is derived from the vertical mass flux (hPa) which is generated by the semi-Lagrangian transport subroutine *tpcore7m.f*. It is divided by the time step length and stored in $hPa\ sec^{-1}$. The values are then time-averaged over the interval between history tape writes. Determination of the vertical wind speed is left to the user. Temperature history files are provided for convenience of post-processing, particularly for performing analysis on isentropic surfaces.

Prior to 13 May 1999, the execution date, the experiment beginning time and date, and the time stamp were represented in the form yymmdd.ff in header words 3, 4, and 6 respectively. The year number was represented by two digits. The updated representation uses four digits. So two 32-bit words instead of one are needed to retain second-level precision in time stamping. Words 3, 4, and 5 now contain the month, day, and day fraction (mmdd.ff) and words 24, 25, and 26 contain the respective year numbers, enabling unambiguous determination of the century. The IDL postprocessor, *pctmr3d.pro* is backwards compatible and returns the correct date and time for both newer and older experiments. However the year must lie between 1950 and 2050 in the older experiments.

4.3 Mass flux divergence history files

The transport routine, *tpcore7m.f*, calculates the local change in the constituent mixing ratio at each time step by summing the three orthogonal components of the mass flux divergence. For access to the components of the mass flux divergence the user must set the script variable `fluxdiv` to `.true.`. The PCTM will then write a mass flux divergence history files for each constituent whose history file write flag is set to `.true.`. These files contain the longitudinal, latitudinal, and total mass flux divergences summed over the time interval between history file writes. The file structure mimics that described for the constituent history files (previous Section 4.2) except that the header record is followed by three arrays of size (`long,lat+1,levels`) instead of one array.

The result is dimensionless and constitutes the change in the mixing ratio since the last history file write due to (1) the longitudinal component of, (2) the latitudinal component of, and (3) all three components of the mass flux divergence. The PCTM does not write the vertical component to the file because of operational considerations. Its computation is left as an exercise to the user. In addition, the change in mixing ratio between history file writes due to the parameterized chemistry can be found by subtracting the total mass flux divergence from the difference of the mixing ratios at the current and previous history file writes.

Mass conservation is problematic in the terrain-following domain for off-line integrations and may adversely affect budget studies focused on the troposphere. Please see Section 10.3 before activating the mass flux divergences switch.

4.4 Source dynamics

The model requires periodic updates to the surface pressure, the zonal and meridional wind components, and the temperature. Of course, the winds participate directly in the transport subroutine, *tpcore7m.f.* The surface pressure is required for the vertical discretization and the temperature is needed to calculate the local number density. A limitation of the current version is that the updates *cannot* be aperiodic. Further, each source dynamics file is required to contain the same number of updates, or "observations." Each observation must have an accompanying time stamp from which the date can be derived.

The functional aspects of obtaining the winds and assigning the logical unit numbers are detailed in Section 3.2. We are limited to 15 files for each submission. As the model executes, each of the required files are read on an as-needed basis. In the example provided, winds from the Data Assimilation Office are read by `subroutine readwinds`. Note that 32-bit words are expected. This is because of the `-F null -N ieee` option on the Cray file assignment statements in the winds generation program.

4.5 Cumulus transport parameterization

A cumulus transport algorithm, subroutine `cldmx`, can be utilized to to incorporate transport by horizontally subgrid-scale clouds (*Lin*, 1996). To activate this section of the code, the user must set `$cloud` switch to `.true.` in the operations script (Section 6.1). `cldmx` requires both the cloud mass flux and detrainment from the Data Assimilation Office GEOS model. Cloud mass flux and detrainment are defined at layer edges and there are no clouds at the ground, so the arrays are declared with length **levels** in the third dimension. Their units are not $kg\ m\ s^{-1}$ as stated on GEOS header records. They are instead $kg\ m^{-2}$ per relaxed Arakawa-Schubert scheme (RAS) time step length. Since the RAS time step is ten minutes, the quantities that are read from disk must be divided by 600 before they are used.

Accessing cloud mass flux and detrainment datasets is similar to accessing the dynamics, and a $windfinder$ which locates and transfers both types of datasets is available.

The arrays **AP** and **BP** are required to determine the local pressure within *cldmx*. The general form for pressure in *cldmx* is

$$p = ap_t + bp_x$$

where p_t is the model lid pressure and p_x is arbitrary, but is stated by on-line documentation as the sum of the surface pressure plus a constant. The isobaric portion of the domain is thus completely represented by

$$a = \frac{p}{p_t}$$

and $b = 0$. In the terrain-following portion of the domain, the local pressure is defined by Equation 4. If we let $p_x = p_s - p_i$, then

$$a = \frac{p_i}{p_t}$$

and $b = \sigma$.

4.6 Startup and Restart files

Excepting perhaps the constituent initial condition file, startup and restart files are identically structured, with restart files serving as startup files for the subsequent sequential submittal. Restart files are written by `subroutine writerestart` and startup files are read by `subroutine readrestart`. Each file consists two records. The first record consists of the experiment number, the data array dimensions, the number of constituents, and the model elapsed time in seconds. The second record is the set of concatenated constituent arrays. For example

```
real const(long,0:lat,levels,ncon)
 ⋮
read (lun) nexp,long,latplus1,levels,ncon,time
read (lun) const
```

The logical unit numbers, provided by the main program, are 95 for reading the startup file and 96 for writing the restart file.

In the default version of the PCTM `subroutine startup` reads two three-dimensional constituent arrays which establish the global initial condition for odd-oxygen, O_x, and odd nitrogen, NO_y. Since the file was written using IDL floating point numbers, an inline conversion of 32- to 64-bit words is required. The initial condition file is read from logical unit number 95 and must have a conformal file name (see Table 1) so that its existence is detected by the operations script. The example of Section 3.1 has the following test:

```
fspec=null
if [ $rstar = '.true.' ] ; then fspec=f77 ; fi.
```

This test must be deleted or restructured if the initial condition has file format `f77`.

The method of establishing the constituent initial conditions is largely left to the user. Other initial conditions may be easily incorporated by writing alternate versions of `subroutine startup`. See Section 9 for details on source code alteration and incorporation. In the special case where the initial condition is analytic and no data is to be read, modifications to the operations script will be necessary. First, the assignment of the logical unit number changes from

```
assign -a $startup -F $fspec fort.95
```

to

```
if [ $iter8n -gt 1 ] ; then assign -a $startup ...
  ... -F f77 fort.95  ; fi
```

This negates assigning a logical unit number at initialization. Second, we will not need try to locate an initial condition file at initialization. Hence, the following can be wrapped around the section which finds the startup file:

```
if [ $rflag -eq 1 ] ; then
fi.
```

The intervening statements will be executed only when the restart flag is raised.

4.7 Parameterized chemistry

The parameterized chemistry in the default version is representative of the issues that the PCTM is designed to address. The PCTM modelling home page on the Code 916 cluster provides a few examples of results. The default version is set up for two constituents: bromoform, $CHBr_3$, and carbon dioxide, CO_2. Each is subject to photochemical loss chiefly in the stratosphere and each is a surface source gas. A detailed description of the chemistry for both constituents is provided by

> */home/nielsen/ctm/ffsl/pchem/chbr3/doc/chbr3.ps*

and will not be repeated here. Three files are required:

> */misc/ctm02/nielsen/p301.pc.dat*
> */home/nielsen/dao/sfccod.144091*
> */misc/ctm02/nielsen/p301.co2bc.dat*.

The first supplies the P and L in Equation 3. The second contains a code which describes the surface characteristics for each ground-level grid box. It is used to provide a mask for the oceans, the origin of most of the $CHBr_3$ surface emission. The last file provides a latitude- and time-dependent CO_2 boundary condition. The routines which read and process this information are `pchem`, `chemparm`, and `co2_interp`. Parameterized chemistry is performed following the advection and cloud mass flux and detrainment.

5 Compilation

A *makefile* compiles the PCTM. An example can be found at

> */home/nielsen/ctm/ffsl/pchem/default/makefile_default*.

Makefiles allow the user to specify a set of rules for building the program. They determine what needs to be recompiled when changes are made to the code modules and avoid recompiling the entire program when only one or two lines of code are changed. Thus, they provide great efficiency especially when debugging.

The first few declarations in the makefile are for the directories which contain the source codes. The intention is to refer to them by variable name, which saves on keystrokes, makes it much easier to plug in and pull out program modules, and facilitates modification of the makefile when code moves to new directories. Next the storage location for the executable is named. This *must* be on the directory `$workdir`. `CFLAGS`n are compile options. Two are included. Most of the program modules are compiled with the f90 options

> `-64 -i8 -r8 -static -O3`.

But the module which includes a call to the intrinsic function `DATE_AND_TIME` must be compiled without the `-i8 -r8` options.

Following the two compiler options lines is the list of rules which are followed when generating the executable. First, the dependency for the executable is listed

```
$EXEC: $(OBJECTS)
   f90 $(CFLAGS2) $(OBJECTS) -o $(EXEC).
```

The executable requires existence of each file in the list of object files. Next the dependency for each object file (external libraries excepted) is listed. For example

```
mainpchem.o: $(PCHEMDIR)/mainpchem.f
        f90 $(CFLAGS1) -c $(PCHEMDIR)/mainpchem.f
```

Each member of the dependency list must have a one-to-one correspondence with each member of the above object list. If, for example, *make* finds that `mainpchem.f` has been altered since the creation date of `$EXEC`, it invokes the chain of dependency by compiling `mainpchem.f` and creating a new `mainpchem.o` file. The dependencies are checked for each member of the list. Compilation occurs only if a `.o` file is missing or a `.f` file has been altered. Once all dependencies have been recognized, the target `$EXEC` is regenerated. Note that `modules.o` must be first in each of the lists because the compiler generates several `.mod` object files which are needed to satisfy the `use` declarations in each of the subroutines.

To use the *makefile*, first obtain a copy for yourself. Then set the pathname `WORKDIR` to your choice for the working directory. This must be the same as `$workdir` in the operations script. From the directory on which your copy of the *makefile* resides, issue the command

```
make -f makefile_default.
```

The *make* utitlity creates several `.mod` and `.o` in this directory. Do not remove these files if you want to take advantage of the time-saving features of *make*. The executable is created on `WORKDIR`, and if no changes are made in dependencies list, the default version of the PCTM is now ready to use.

6 Setting the Parameters

Run time parameters are declared in two separate files. Parameters which control execution are established within the operations script, *run.default* for example. Internal parameters which for the most part determine array sizes are left to the source code file *modules.f*. Each is described in the following subsections.

6.1 Operations script

The first several lines of the script list control parameters. These must be thoroughly reviewed each time the model is executed. For those familiar with older versions of the code, this structure functionally replaces the input file *parameters.input*. In fact, we now use the script to internally compute several items, making automatic resubmittal possible. The portion of the script which digests the user-supplied declarations is discussed in Section 3.1. Table 3 lists the input parameters required by the script.

Additional notes for Table 3 follow. The item number matches the reference from the "Note" column in the table:

1.0. `$workdir` is generally a data disk. Do not use */home*.

1.1. Active constituents, if any, must be listed first, and their number must be equivalent to the designation for `nactive` in `modules.f`. Inert compounds (tracers), if any, must be listed following the active constituents, and their number must be equivalent to `ntracers` in `modules.f`. See also Section 6.2. Each constituent must have an history file write flag.

1.2. Each run of the sequence will have the same length.

1.3. Ending times are computed internally.

1.4. If beginning date and time and initial condition date and time are equal, then the script will tell the model that the first time step will not be a restart.

1.5. $n\Delta t$ must equal 3600, where n is a positive integer.

1.6. χ^* is obtained by calling *tpcore7m* `ichemfac` times, each with a time step of length `tdt`. The chemistry is then updated to complete the prognostication of $\chi(t+\Delta t)$. That is, Δt in Equation 1 is actually `ichemfac * tdt`. The length of the dynamics time step is always less than or equal to the length of the chemistry time step.

1.7. `dthist` is specified in integer hours. If the value is less than or equal to zero, then the output interval will be each dynamics time step, `tdt` seconds.

1.8. Normally each successive set of restart records overwrites the previous set. If a negative number is entered, then successive records are concatenated. This has the potential of generating a large file.

1.9. Each source file (which contains surface pressure, horizontal wind components, and temperatures) must contain the same number of observations, spaced at equivalent intervals.

1.10. Affects only those constituents which are active. See Note 1.

1.11. If `.true.` then vertical index in history file arrays increases toward higher pressure, as in the model. If `.false.` then the vertical index is reversed on the history tape arrays (only) into a ground-to-lid configuration.

1.12. See *tpcore7m.f*, where the differences between "A" and "C" grids are described.

Table 3: Run-time parameters			
Note	Name	Example	Description
1.0	`workdir`	*/misc ...*	Working directory.
	`executable`	*pchem.default*	Name of executable on `$workdir`.
	`dir4wind`	*/misc ...*	Directory on which the windfinder resides.
	`execdir`	*/home ...*	Directory on which *phonedirac* resides.
	`clockdir`	*/home ...*	Directory on which *caladvance* resides.
	`restartdir`	*/u3/y3jen/...*	Unitree directory from which a restart file may be obtained.
	`archivedir`	*/u3/y3jen/...*	Unitree directory upon which history and restart files are permanently archived.
	`numcon`	2	Number of constituents, must be ≤ 9.
	`testmode`	0	Test mode indicator. See Section 10.1.
1.1	`speciesname`$[n]$	n2o	Species name for constituent n. Lower case, maximum 8 characters.
	`saveflag`$[n]$	.true.	History file write switch for `speciesname`$[n]$.
	`expno`	34	1-, 2-, or 3-digit experiment number.
1.2	`numruns`	3	Number of sequential submissions requested.
1.2	`runlength`	10	Length in days of each of the `numruns` submittals.
1.3	`byymmdd, bhh`	19970412 0	Date and hour (UTC) to begin this experiment.
1.4	`icyymmdd0, ichh`	19970410 12	Date and hour (UTC) of initial condition.
	`nexp0`	295	Experiment number.
1.5	`tdt`	900.00	Dynamical time step Δt in seconds.
1.6	`ichemfac`	1	Number of dynamics steps taken to establish intermediate constituent values, χ^*.
1.7	`dthist`	6	History file write interval in hours.
1.8	`dtrestart`	96	Restart file write interval in hours.
1.9	`sourcedays`	10	Number of days in each source winds dataset.
1.9	`dtwind`	6.00	Time interval in hours between successive winds observations.
	`transport`	.true.	Switch to enable calls to transport algorithm.
1.10	`chemistry`	.true.	Switch to enable photochemistry for active constituents.
	`cloud`	.false.	Switch to enable parameterization of subgrid-scale cumulus transport.
	`fluxdiv`	.false.	Compute and store components of the mass flux divergence.
1.11	`histtd`	.true .	History file vertical orientation switch.
1.12	`igd`	0	Grid selection for *tpcore7m.f.* 0='A' grid 1='C' grid.
	`storetk`	.false.	Create a history file for temperature (at unit 69).
	`storew`	.false.	Create a history file for vertical wind speed (at unit 68).
	`storepv`	.false.	Create a history file for potential vorticity (at unit 67).
	`storepbl`	.false.	Not used in the default version.
Note number precedes items which are explained in more detail in the text.			

Table 4: User-determined parameters in modules.f			
Note	Name	Example	Description
	`kassim`	46	Number of vertical layers in source winds datasets.
	`ktrop`	6	Number of layers in the terrain-following portion of the PCTM. See Equation 5.
	`kstrat`	31	Number of layers in the isobaric portion of the PCTM. See Equation 6.
	`long`	144	Number of equally spaced longitudes on each latitude circle.
	`firstlon`	-180.00	Longitude of first grid point on each latitude circle, generally 0 or -180.
2.1	`lat`	90	Number of latitude cells, or equivalently, number of grid points on each meridian minus 1.
2.2	`ncon`	2	Total number of constituents, both active and inert.
2.2	`nactive`	1	Number of active constituents.
2.2	`ntracers`	0	Number of inert constituents.
Note number precedes items which are explained in more detail in the text.			

6.2 modules.f

Here the user must declare several parameters which are related to array sizes, number of constituents, and model layer pressures. The list is given in Table 4. This file must be thoroughly reviewed when the user changes the resolution of the model or adds or deletes constituents. Previous versions required the user to declare the lid pressure (`ptop`), the terrain-following to isobaric interface pressure (`pint`), and the sigma coordinate of the layer interfaces. The default version now defines each of these parameters in the vertical discretization definition subroutine.

Additional notes for Table 4 follow. The item number matches the reference from the "Note" column in the table:

2.1. Winds, pressures, and temperatures are colocated and lie at the edges of the latitude cells, or at `lat`+1 latitudes. Their arrays are thus dimensioned `(0:lat)` in the north-south direction.

2.2. If `ncon`≠`nactive`+`ntracers`, the PCTM will terminate.

7 Adding or Deleting Constituents

The PCTM has the ability to transport inert tracers along with photochemically active constituents. However, the model expects the active constituents, when they are included, to be listed first, ahead of the inert tracers in the execution script (See Section 6.1). The user tells the model which constituents are active and which are inert in *modules.f* (See Section 6.2). Experiments which have no active compounds are allowed, as are experiments with no inert compounds. The chemistry activation switch, `chemistry`, affects only the active constituents. If it is turned off, then even the active constituents are simply transported.

Adding and deleting constituents is now considerably easier than in earlier versions of the model. The script does much of the work. But the user must carefully perform the following steps to add or delete constituents.

7.1 Operations script

Near the beginning of the list of user-defined parameters, declare the number of constituents, their names, and a history file write switch for each constituent. At this writing we are limited to nine (9) constituents and each name is limited to eight (8) characters. The character strings entered here are used to discretely identify the history file for each respective constituent. Thus we have usually adopted lower case. For example, if production and loss rates are available for N_2O and CH_4 and one tracer is also desired, then set

```
numcon=3
speciesname[1]=n2o       saveflag[1]=.true.
speciesname[2]=ch4       saveflag[2]=.true.
speciesname[3]=tracer    saveflag[3]=.true.
```

The numerals in brackets are array element specifiers for the shell. The numerals must appear in order $1, 2, \ldots, n$ where $n =$`numcon`. All active constituents *must* be listed before any inert constituents.

7.2 modules.f

Here we (again) tell the model how many constituents are included in the integration. Additionally, we designate which ones are active and which ones are inert. The three pertinent lines are in *module dimensions*:

```
parameter(ncon=3)
parameter(nactive=2)
parameter(ntracers=1)
```

The value assigned to `ncon` must equal that assigned to `$numcon` in the execution script. The model tests for this condition. The value assigned to `nactive` must equal the number of active constituents. The acceptable range is $0 \leq$ `nactive` $\leq$ `ncon`. The value assigned to `ntracers` must equal the number of inert constituents. The acceptable range is $0 \leq$ `ntracers` $\leq$ `ncon`. And `nactive` + `ntracers` must equal `ncon`. The model also tests for this condition.

8 Running the model

The PCTM can be executed only on those machines which have a Fortran 90 compiler and which have an unix operating system with the *assign* command. For testing the example, four files must be copied into your own directory. They are the makefile for compilation, the operations script, the selector script, and the wind finder. Respectively,

/home/nielsen/ctm/ffsl/pchem/default/makefile_default
/home/nielsen/ctm/ffsl/pchem/default/run.default
/home/nielsen/ctm/ffsl/pchem/default/selector_strat
/home/nielsen/ctm/ffsl/pchem/default/windfinder_strat.

These scripts are detailed in Chapter 5, Section 3.1, and Section 3.2. The working directory name must be uniformly declared in each script. For the makefile, declare the working directory by setting

```
WORKDIR=/misc/abc0n/username
```

where `abc0n` is the disk name and `username` is your login identification on the Code 916 unix cluster. In the operations script the following three run-time parameters are unique to each user and experiment and must be provided:

```
workdir=/misc/abc0n/username
restartdir=/u3/y3jen/ctm/example/exp
archivedir=/u9/xnuser.
```

Otherwise, the rest of the run time parameters are set for a short, single integration experiment. The restart and archive directories reside on Unitree, so *you must have an authorized account on Unitree and/or helios1* before proceding. Here, `/u9/xnuser/` is your home directory on Unitree. The selector and the wind finder are applicable only to STRAT assimilation archive.

Before compilation, review Chapter 5. Then from the directory in which *your* makefile resides, enter `make -f makefile_default` on the command line. The object will be stored at `$EXEC` on the working directory. Initial conditions and production and loss rates exist for the example and may be transferred from Untree. Review Sections 3.1 and 3.2, and set the run-time parameters. In production mode, the PCTM is best executed on the batch machine via the following commands:

```
at now + 1 minute
run.default > default.log
<Ctrl + d>
```

The contents of *default.log* detail the script operations. This file is created in the directory from which the batch command is executed. Do not confuse this file with `$logfile`, which contains standard output from the PCTM and is created on `$workdir`. See the manual page for `at`. An example of the batch log can be found on Unitree at

/u3/y3jen/ctm/example/exp999/default.log.

In test mode, the PCTM is executed by entering

```
run.default
```

on the command line. Information provided by the script, which goes into *default.log* when in production mode, is instead scrolled to the screen along with standard output from the PCTM.

After the PCTM completes execution, the operations scripts removes its temporary files from `$workdir`. When execution terminates abnormally, whether in production or test mode, the user is advised to remove these files before resubmitting.

9 Altering the source code

This section outlines the procedures for making changes to the source code and integrating the new code into the PCTM. While the user need not be concerned with maintenance of most of the low-level source code modules, the parameterized chemistry and the configuration of the startup files, for example, will need to be modified to suit individual experiments. The use of makefiles (see Section 5) reduces new code implementation to a few simple steps so that the user may devote most of his or her energy to programming the subroutines. Those steps are:

- Design and code new and/or replacement subroutines.
- Replace, add, or delete the respective object file names from the list of object files in the makefile.
- Alter the respective dependencies in the makefile, preserving the one-to-one correspondence of the object files with their dependencies.
- If necessary rename the executable, `$EXEC`, in the makefile.
- Compile and run.

Thus new code incorporation is essentially a book-keeping task within the makefile where the new object reference and its dependency are substituted in place of those that already exist. A unique makefile may be required for each experiment. When replacing source code modules, the new subroutines must retain the same call list parameters as their predecessors unless the calling routine is also modified. Upon completion of the compile step, the makefile will overwrite an existing executable if the name `$EXEC` is unchanged. Though it is not recommended, changes to the structure of external files or the methods by which they are accessed must be accompanied by appropriate alterations to the operations script (see Section 3).

10 Programming notes

The Appendix contains variable lists for the main program and for the modules. Individual subroutines are listed in the Appendix, Section 11.3, each with a brief statement of purpose.

10.1 Test modes

The script variable `$testmode` allows debugging without invoking the file maintenance (scrubbing) characteristics of the production mode. While in test mode, the PCTM has the following characteristics:

- Standard output is sent to the screen.
- Local copies of the history files, the winds source files, and the startup file are *not* removed upon end of execution.
- The script variable `$numruns` is reset to `1`.
- Preselection of winds files is turned off.
- The command file for file transfers to Unitree is echoed to the screen.

Production mode is initiated when `$testmode` is set to zero. Test modes are available at two levels. When `testmode=1`, the features listed above are implemented, plus the archival section of the script is executed. This allows the user to verify the selection and naming of `$archivedir` and of the startup, history, and job log files, and to make sure the file transfers succeed. Test mode 2, `testmode=2`, is the same as test mode 1 except the archival functions are *not* tested. While testing, the script prints the names of the files it will remove if production mode is chosen with the same set of user-defined script variables.

10.2 Vertical discretization

In general, the PCTM operates with vertical index number `1` at the the model's lid (lowest pressure) and vertical index `levels` near or at the planetary surface. This conforms to required orientation of all arrays upon calls to both transport routines, *tpcore7m* and *cldmx*. The local pressures are obtained from Equations 4 and 5. Many, but not all, source dynamics files are archived with their vertical indicies reversed. Therefore, the user must be aware of the vertical arrangement of the source dynamics data when designing experiments. Note that `subroutine zflip` can be used to reverse arrays in the vertical direction.

While we have imposed a uniform "lid-to-ground" filesystem within the PCTM, it is expected that many users will need to accomodate an existing array of "ground-to-lid" post-processors. This is accomplished by setting the switch `histtd` to `.false.` in the execution script. When this is done, `subroutine writehistory` will flip the arrays in the vertical before writing the records. Please see comments in Section 4.2, where the history files are discussed.

10.3 Mass conservation

Early versions of the transport routine, *tpcore*, do not conserve the mass of an inert tracer in off-line simulations with topography and the hybrid σ/p vertical coordinate unless the surface pressure is available from the source dynamics at each time step. Nonconservation worsens as the pressure which marks the boundary between the terrain-following and isobaric domains, the interface pressure p_i (See Equation 4), falls. Placing p_i well within the troposphere, but making sure that it is not near to or greater than the minimum surface pressure in the source dynamics dataset, and using as few terrain-following layers as is prudent is recommended to minumize mass nonconservation. The default version of the PCTM places the interface pressure at 337 hPa and initializes six terrain-following layers. The user is encouraged to correspond with Dr. S.-J. Lin for further details.

Version number seven of *tpcore* displays significantly improved mass conservation. However, it uses the so-called η vertical coordinate instead of σ. A version of the PCTM which is written for the η vertical coordinate is in the directory

/home/nielsen/ctm/ffsl/pchem/eta

but has not yet been made the default. However the default version uses the η coordinate version of *tpcore*. The η version of the PCTM is required if dynamics from the Finite Volume Community Climate Model (FVCCM) will be used to perform the integration.

10.4 Arrays AP and BP for cldmx and tpcore7

The two subroutines *cldmx.f* and *tpcore7* were written by Dr. S.-J. Lin in conformance with the η vertical coordinate that is used in the new generation DAO GCM, the so-called Finite Volume Community Climate Model (FVCCM). Each of these subroutines require two arrays of length `levels+1`, `AP` and `BP`, which are used to define the layer edge pressure. In general the local pressure is defined within these subroutines by

$$p = ap_t + bp_x \tag{6}$$

where p_t is the model lid pressure and p_x is the real pressure minus an arbitrary constant. With the hybrid-σ coordinate versions of the PCTM, calls to *tpcore* define $p_x = p_s - p_i$ where p_s is the surface pressure and p_i is the layer edge pressure at the interface between the terrain-following region and the isobaric region. Hence, in the isobaric region, $b = 0$ and

$$a = p/p_t.$$

In the terrain following region, with Equation 4 as a guide, $b = \sigma$ and

$$a = p_i/p_t.$$

10.5 Memory faults in the SGIs

"Segmetation faults" have occurred on the Code 916 SGI cluster and on the DAO's DEC-α workstation, *molotov* when running the PCTM with fine resolution, with several constituents, or with both. While the source of the fault is not certainly identified, an apparent work around is to compile the code with the `-static` option. The user is invited to investigate any compile problems and the vast array of optimization options.

11 References

Douglass, A. R., C. H. Jackman, and R. S. Stolarski. Comparison of model results transporting the odd nitrogen family with results transporting odd nitrogen species. *J. Geophys. Res.*, 94, 9862-9872, 1989.

Lin, S.-J. Description of the parameterization of cumulus transport in the 3D Goddard chemistry transport model, *Internal memo.*, 5 pages, 1996.

Lin, S.-J., and R. B. Rood. Multidimensional flux-form semi-Lagrangian transport schemes. *Mon. Wea. Rev.*, 124, 2046-2070, 1996.

McRea, G. J., W. R. Goodin, and J. H. Seinfeld. Numerical solution of the atmospheric diffusion equation for chemically reacting flows. *J. Comput. Phys.*, 45, 1-42, 1982.

Rood, R. B., A. R. Douglass, J. A. Kaye, M. A. Geller, C. Yuechen, D. J. Allen, E. M. Larson, E. R. Nash, and J. E. Nielsen. Three-dimensional simulations of wintertime ozone variability in the lower stratosphere. *J. Geophys. Res.*, 96, 5055-5071, 1991.

Rood, R. B. Numerical advection algorithms and their role in atmospheric transport and chemistry models. *Rev. Geophys.*, 25, 71-100,1987.

Tacaks, L. L., A. Molod, and T. Wang. Documentation of the Goddard Earth Observing System (GEOS) general circulation model: Volume 1. *NASA Technical Memorandum 104606, Vol. 1*, 1994.

12 Appendix

12.1 Variable list for modules.f

Name	Description
ap, bp	Vertical coordinate arrays for *tpcore*. See Section 10.4.
avecos	Diurnally averaged cosine of solar zenith angle.
begyrs	Year of initial condition.
begmon	Month number of initial condition.
begday	Day of month of initial condition.
beghrs	Hour of initial condition.
begmin	Minute of initial condition.
begsec	Second of initial condition.
botzmann	Boltzmann's constant, JK^{-1}.
cellarea	Surface area (m^2) of bottom of each grid box.
coast	Flag to indicate whether surface grid box is coastal (1) or not (0).
co2bc	Array to hold CO_2 surface exchange.
co2ncep	CO_2 surface exchange from CMDL.
co2weeks, co2yrs	Number of weeks and years on the CO_2 surface exchange dataset.
csubp	Specific heat capacity of dry air at constant pressure, $Jkg^{-1}K^{-1}$.
decrad	Apparent solar declination in radians.
firstlon	First longitude on each latitude circle.
gravity	Acceleration of gravity, $msec^{-2}$.
hrang	Hour angle of the sun.
kstrat	Number of layers in *pressure* region.
levels	Number of layers, ktrop+kstrat.
ktrop	Number of layers in σ,terrain following, region. See Equs. 5 and 6.
lat	Number of latitudes for chemistry model.
lenhdr	Number of words in the history tape header record.
levels	Total number of layers.
long	Number of longitudes.
loss	Loss rates for active constituents, L in Equation 1.
metday	Model elapsed time. *dd* of *dd* d *hh* h *mm* m *ss* s.
methrs	Model elapsed time. *hh* of *dd* d *hh* h *mm* m *ss* s.
metmin	Model elapsed time. *mm* of *dd* d *hh* h *mm* m *ss* s.
metsec	Model elapsed time. *ss* of *dd* d *hh* h *mm* m *ss* s.

Variable list for modules.f, continued.

Name	Description
`monthdays`	Number of days in each month. Note that the entry for February is continuously updated by the model's clock.
`monthname`	Three-character month name.
`mwtair`	Average molecular weight of dry air, $kgkmol^{-1}$.
`mwtco2`	Average molecular weight of CO_2, $kgkmol^{-1}$.
`nactive`	Number of active constituents.
`ncon`	Number of constituents, active and inert.
`nowyrs`	Current year.
`nowmon`	Current month number.
`nowday`	Current day of month.
`nowhrs`	Current hour.
`nowmin`	Current minute.
`nowsec`	Current second.
`nsuba`	Avogadro's number, $kmol^{-1}$.
`ntapes`	A number greater than or equal to the the expected number of source dynamics datasets. Used for dimensioning.
`ntracers`	Number of inert constituents.
`omega`	Angular speed of rotation of planet, sec^{-1}.
`pi`	Ratio of the circumference of a circle to its diameter.
`pint`	σ-to-p interface pressure, hPa. See Equation 5.
`press`	Approximate layer mean pressure, hPa.
`prslay`	Layer edge pressures, hPa.
`prod`	Production rates for active constituents, P in Equation 1.
`ptop`	Model lid pressure, hPa.
`radius`	Planetary radius, km.
`rau`	Distance to the sun in astronomical units.
`rgas`	Gas constant for dry air, $Jkg^{-1}K^{-1}$.
`sfccod`	Integer indicating surface characteristics [1=ocean, 2=land, 3=ice].
`sgint`	Layer interface vertical coordinate σ_i
`sigma`	Layer vertical coordinate σ.

12.2 Main program variable list

Name	Description
`chemistry`	Switch to enable photochemistry for active constituents.
`const`	Constituent mixing ratios.
`coscen`	$\cos(\phi)$, $\phi =$`rlatcen`$(\pi/180)$.
`cosedge`	$\cos(\phi)$, $\phi =$`latedge`$(\pi/180)$.
`dayn0`	Initial condition date, ddmmyy.ff.
`dthist`	Time interval between history file writes.
`dtrestart`	Restart file write interval.
`dtwind`	Time interval in hours between successive winds observations.
`fill`	Switch to enable filling of negative values in *tpcore*.
`fluxdiv`	Switch to enable calculation and storage of the components of the mass flux divergence.
`head`	History file header record. See Table 2.
`ichemfac`	Number of dynamics steps taken to establish intermediate constituent values, χ^*.
`iord`	Option for controlling longitudinal aspects of transport. See *tpcore*.
`iskip1`	Number of wind observations which are skipped in order to position source winds file at correct starting time.
`ist1`	Number of required observations from each winds source file.
`itape1`	Winds dataset number which contains the first required observation.
`iter8n`	Cumulative iteration number, time step length `tdt`.
`jord`	Option for controlling latitudinal aspects of transport. See *tpcore*.
`kel1, kel2`	Temperature from previous and next, respectively, observations.
`kord`	Option for controlling vertical aspects of transport. See *tpcore*.
`latcen`	Latitude (radians) of center of grid box.

Main program variable list, continued.

Name	Description
`latedge`	Latitudes (degrees) of data and edge of grid box.
`lun`	Logical unit numbers for winds datasets, except `lun0`.
`lun0`	Logical unit numbers for first required winds dataset.
`lunfluxdiv`	Logical unit numbers for history tapes of the components of the mass flux divergence.
`lunhistory`	Logical unit numbers for constituent history tapes.
`lunloss`	Logical unit number for loss rates.
`lunprod`	Logical unit number for production rates.
`lunrs1`	Logical unit number for startup file.
`lunsave1`	Logical unit number for restart file.
`m`	Number density for air.
`metstr`	Character string for model elapsed time in the form *ddd* d *hh* h *mm* m *ss* s.
`mfct`	Switch to perform multi-dimensional flux correction term (*tpcore*).
`ndt`	Time step length for *tpcore*. Integer of *tdt*
`nexp`	Experiment number.
`nsteps`	Number of "observations" on each source dynamics file.
`ntshis`	Number of time steps between successive history file writes.
`ntsrst`	Number of time steps between successive restart file writes.
`ntstrc`	Number of time steps between successive dynamics file reads.
`ps1, ps2`	Surface pressure from previous and next, respectively, "observations."
`psminusc0`	Surface pressure at current time for *tpcore*.
`psminusc1`	Surface pressure at mid-level time for input to *tpcore*. Overwritten with predicted surface pressure on output.
`rewres`	Switch which tells *writerestart* to rewind the restart file.
`restart`	Restart flag.
`sourcedays`	Number of days contained in each source winds dataset.
`species`	Switch for each constituent which enables history file writes.

Main program variable list, continued.

Name	Description
`speciesname`	Constituent identification, `character(len=8)`.
`storetk`	Create a history file for temperature (at unit 69).
`storew`	Create a history file for vertical wind (at unit 68).
`tapes2read`	Number of source dynamics files needed to execute this sequential run.
`tdt`	Time step applied to transport algorithm, seconds.
`testmode`	Test mode indicator. See Section 10.1.
`time`	Model elapsed time, seconds.
`time2`	Time at center of *tpcore* time step.
`transport`	Switch to enable use of transport algorithm.
`twind1`	Model elapsed time at "previous" wind observation.
`twind2`	Model elapsed time at "next" wind observation.
`u1, u2`	Zonal wind component from previous and next, respectively, observations.
`uflx,vflx, tflx`	Longitudinal, latitudinal, and total mass flux divergence, cumulative between history file writes.
`umax`	Estimate of the maximum U-wind speed (m/s) for *tpcore*.
`utcstr`	Character string for time of day in the form hh:mm:ss UTC dd Mon yyyy.
`v1, v2`	Meridional wind component from previous and next, respectively, observations.
`w`	Vertical mass flux generated by *tpcore* versions 4.0 and greater.
`winddate`	Date of wind observation, yymmdd.ff.

12.3 List of subroutines

Name	Function
`ab4sigma`	Generate arrays **AP** and **BP** for *tpcore7m*. See Section 10.4.
`calstring`	Assemble the string hh:mm:ss UTC Dow dd Mon yyyy from the given year number, month number, day of month, hour of day, minutes, and seconds.
`chemparm`	Read parameterized chemistry files and interpolate to current day of year.
`cldmx`	Algorithm for subgrid-scale cumulus transport.
`co2_interp`	Generate the current surface exchange for CO_2.
`dates`	Reads system clock and returns the current calendar date as a real number in the form yymmdd.ff.
`dayofweek`	Returns day of week as a three-character string.
`dhmsstring`	Assemble a string of the form *ddd* d *hh* h *mm* m *ss* s from the given days, hours, minutes, and seconds.
`epvgen`	Calculate Ertel's potential vorticity.
`header`	Fill the history file header record. See Table 2.
`initialize`	Obtain initial number density, initialize constituent values, and verify time stamp on source winds dataset.
`julday`	Calculate Julian day. 12:00 UTC 30 Dec 1899 is 2415019.
`mass3d`	From a given mixing ratio, determine the total (global) mass of each of `ncon` constituents.
`numdens`	Calculate the local number density for air.
`pchem`	Driver subroutine for parameterized chemistry.
`polave`	Average polar values in a three-dimensional array.
`radslr`	Calculate position of sun and related parameters.
`readcloud`	Read the cloud mass flux and detrainment.

List of subroutines, continued.

Name	Function
`readrestart`	Read the startup file at logical unit `lunrs1`.
`readwinds`	Read the horizontal winds, temperatures, and surface pressures from the logical units assigned by the execution script.
`second`	Obtain user, system, and total CPU time in seconds.
`sethoriz`	Initialize the longitude-latitude gridding for edge and center of grid boxes.
`setparams`	Establish the run-time parameters, most of which originate with the execution script. See Section 6.
`setver`	Establish the vertical level-layer discretization.
`startup`	Initialize the constituents to begin an experiment.
`tapecheck`	Determine pertinent information regarding source wind datasets. Warning: Four observations per day is assumed.
`timecheck`	Obtain the current date and time of day given a starting date and an elapsed time.
`timer`	Resource allocation determination subroutine.
`thetagen`	Calculate the potential temperature.
`tpcore`	Transport algorithm.
`tpcoreset`	Establish parameters in call list for *tpcore*.
`tropdef`	Determine if a grid box is in the troposphere (1) or not (0).
`writehistory`	Add to the history file for each constituent.
`writerestart`	Write the restart file, perhaps after rewinding.
`xport`	Driver for the transport algorithm.
`zflip`	Reverse a three-dimensional array in its last dimension.

12.4 Obtaining copies of this document

This document is on-line. To view it on the screen, issue the command

```
ghostview /home/nielsen/ctm/ffsl/pchem/doc/user_guide.ps.
```

To obtain a printed copy, issue the command

```
lpr -PPS_HP2 /home/nielsen/ctm/ffsl/pchem/doc/user_guide.ps.
```

We are continually improving the stucture and performance of the PCTM. We do our best to keep this manual up to date. Users are encouraged to email their comments and suggestions to

eric.nielsen@gsfc.nasa.gov.