

The CCCma Single Column Model Version 1.2

Nikola Tucakov
University of British Columbia
Canada

July 12, 2002

Abstract

Contents

1	Introduction	1
2	Installation	2
2.1	Makefiles	3
3	Program Execution	5
3.1	Running the SCM	5
3.2	Doing Multiple Runs	6
3.3	Setting the Run Parameters	6
3.4	Run Examples	7
4	Implementation	9
4.1	Scripts	10
4.2	Preprocessor	11
4.3	SCM Algorithm	12
4.3.1	Dry Convection Scheme	13
4.3.2	Deep Convection Scheme	14
4.3.3	Turbulence Scheme	14
4.3.4	Radiation Scheme	15
4.3.5	Land Surface Scheme	15
4.3.6	Vertical Turbulent Diffusion Scheme	16
4.4	Parameters	17
4.5	Variables	18
5	Software Overview	19
5.1	Software Components	19
5.1.1	All the components	19
5.1.2	Function Call tree	20
5.1.3	Extra components	20
5.1.4	Changes Made in SCM	21
5.2	NetCDF Files	22
5.2.1	Installing NetCDF	22
5.2.2	About NetCDF	23

5.2.3	UBC SCM NetCDF Data Format	25
6	Input/Output	27
6.1	Input Files	28
6.1.1	All Input Files	28
6.1.2	Preprocessor Files	31
6.2	Output Files	33
6.2.1	All Output Files	33
6.2.2	NetCDF output files	35
7	Plotting	36
8	More References	36

List of Figures

List of Tables

1 Introduction

The CCCma Single Column Model Version 1.2 models a single atmospheric column, using the CCCma's AGCM3 full Physics Package, originally developed at CCCma in Victoria. It was then ported to UBC with some changes in data format and plotting.

The processes modelled are:

- Deep convection
- Dry convective adjustment
- Radiative processes
- Vertical turbulent diffusion
- Land surface scheme and surface heat, moisture and momentum fluxes.

This report contains information on how to install and run the scm at ubc. It also gives the insights to the program algorithm, software components, data format and i/o files used to run the program.

2 Installation

To obtain the model, you have to check it out of CVS. CVS is a tool that maintains a history of a source tree. Multiple users can work on the same projects at the same time using CVS. There is a tutorial that gives details about all of it's commands that are applied to ubcscm: <http://roc.eos.ubc.ca/users/ntucakov/cvstutorial/index.html>

This is how to check the model out:

```
>>> cvs co ubcscm
```

The directory structure will expand as shown below:

```
ubcscm ----| model  --> model program and script files (including RCS directory)
          ----| output --> model output files
          ----| input  --> input files (not here) (should soon be phased out)
          ----| ARM    --> ARM forcing data (included in model dir, might be added later)
          ----| TOGA   --> TOGA forcing data (included in model dir, might be added later)
          ----| GATE   --> GATE forcing data (not here yet)
          ----| doc    --> model documentation
```

Also, you have to check out the code that will generate drivers for model's NetCDF data format.

```
>>> cvs co ccnc      (when using the cpp code) or
>>> cvs co ccncf90   (when using the f90 code)
```

SCM can use either ccnc or ccncf90 library. This depending on the platform where the model is run. When checked out, all the model defaults are set to use f90.

2.1 Makefiles

Table 2.1 summarizes all the makefiles for running the codes on different platforms:

Directory	make file	comiler	run on	comments
cccnc/ cccncf90/	Makefile, makefile	g++	Brant, UBC	NetCDF routines in c++
	Makefile-pgf90	pgf90	Brant, UBC	NetCDF routines in f90
	Makefile-xf90	xf90	CCCma	NetCDF routines in f90
ubscscm/	—	—	Delhousie	NetCDF routines in f90
	makescm-pgf90-cccnc	pgf90	Brant, UBC	SCM, with cccnc
	makescmfdat-pgf90-cccnc	pfg90	Brant, UBC	scmfdat.f, with cccnc
	makescm-pgf90-cccncf90	pgf90	Brant, UBC	SCM, with cccncf90
	makescmfdat-pgf90-cccncf90	pgf90	Brant, UBC	scmfdat.f, with cccncf90
	makescm-xf-cccncf90	xf	CCCma	SCM, with cccncf90
	makescmfdat-xf-cccncf90	xf	CCCma	scmfdat.f, with cccncf90
	—	—	Delhousie	SCM, with cccncf90
—	—	Delhousie	scmfdat.f, with cccncf90	

Note about Compiling the SCM NetCDF Drivers

The cccnc(f90) makefiles' output libraries and module files have to be included by SCM. The way it is set up right now, cccnc(f90) makefiles copy these files into /lib and /mod directories where SCM's makesfiles look in later on. If these directories don't match up on your machine, make sure they exist and are the same.

Making sure the Model can compile

Since there are more than one different make files for different compilers, to run the model with the right compiler on the right system might require some adjustments. Here are some things that might have to be changed.

- which makefile will be run is set in the scmsub.dk (both makescm and makescmfdat)
- in makefiles, SCM_ROOT is set to ".". If you want, you can specify your own working directory

- if scm is run with pgf90/ccnc: comment out the 'use nc_module' line in scmfdat.F and scm_parm.F
- if scm is run with pgf90/ccncf90: ccncf90 has to be compiled with pgf90
- if scm is run with xlf/ccncf90: ccncf90 has to be compiled with xlf90 and both "parmsub-pgf90"s in scmsub.dk should be renamed to "parmsub" (the model contains only parmsub-pgf90 (no parmsub-xlf) and a call to parmsub is defined at CCCma)

By default, UBCSCM is set to run on Brant, UBC using ccncf90.

3 Program Execution

3.1 Running the SCM

Running the SCM can be done either for just one realization, or multiple runs can be performed as an ensemble of up to 100 members, each of them perturbed with data from ARM. To run the model as a single realization, the user must do the following steps:

1. Go to the "model" directory
2. Edit the scm_job file to set up run parameters (more on that below)
3. Launch the SCM with the following command:

```
>>> scm_job RUN_NAME ensemble_member forcing_data subcase
```

Forcing Data

Table 3.1 shows the “forcing data” that the model uses and its “subcase”s.

data	subcase	time	run time	type
ARM_C3	A	2330 UTC 26 June 1997 - 2330 UTC 30 June 1997	4 days	precip
ARM_C3	B	2330 UTC 7 July 1997 - 2330 UTC 12 July 1997	5 days	precip
ARM_C3	C	2330 UTC 12 July 1997 - 2330 UTC 17 July 1997	5 days	precip
ARM_C3	D		14 days	
ARM_C3	R	2330 UTC 18 June 1997 - 2330 UTC 22 June 1997	4 days	dry
ARM_C3	S	2330 UTC 22 June 1997 - 2330 UTC 26 June 1997	4 days	precip
ARM_C3	T	2330 UTC 30 June 1997 - 2330 UTC 3 July 1997	3 days	dry
ARM_C3	U	2330 UTC 4 July 1997 - 2330 UTC 8 July 1997	4 days	dry
ARM_C3	X	2330 UTC 18 June 1997 - 2330 UTC 17 July 1997	29 days	dry and precip
ARM_C3	Y	2330 UTC 2 June 1997 - 2330 UTC 6 July 1997	14 days	
ARM_C1	X		17 days	
TOGA	A		10 days	
TOGA	X	0000 UTC 1 Nov 1992 - 1800UTC 28 Feb 1993	120 days	
GCM	X	Default value		
BOMEX	X	the branch tagged with ubcscm_bomex_branch	2 days	

Example: >>> scm_job myrun 1 ARM_C1 X

3.2 Doing Multiple Runs

This will be done using python script `run_ensemble.py`. The following is the unix script that does it.

The "run_ensembles" script

To run multiple model realizations, a small script wrapper must be used. The "run_ensembles" script contains definitions of the "scm_job" script input parameters which must be provided by the user. The "scm_job" script is then called repeatedly until all ensemble members have been produced. (A second script is then launched ("ensemble_stats") which will compute means and variances of the ensemble members) followed by the python plotting script.

3.3 Setting the Run Parameters

This section describes parameters on the user level. Section 4.4 gives more information on all the rest of the parameters.

The "scm_job" script

This file is the main interface between the user and the SCM. Most of the file contains UNIX variable assignment commands to set up the various parameters used in model execution. For the user's sake, the only section to be concerned about is the MASTER CONDEF PARAMETERS section which contains the following parameters:

compile : Compile the model code with any new changes (using the "make" utility).

run : run the current executable

plotts : plot time series from model output (generate a web page)

plotpr : plot profiles from model output

frcdat : use field campaign observational data or GCM data for lateral and bottom forcing

class : use CLASS module for computing surface fluxes (if "off", use prescribed fluxes)

debug : compile with the "-g" debugging option and launch the xldb debugger

Another section useful to the user is the FORCING DATA PARAMETERS section where nudging can be used ("nudge" parameter).

Note that, like in all CCCma's submission jobs, condef parameters are off by default unless explicitly turned on.

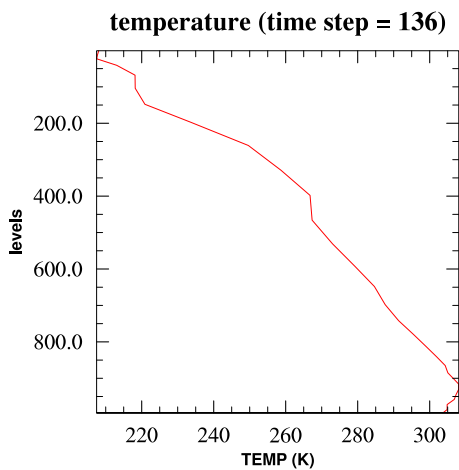
When launched, the scm_job script will (if all master condef parameters are set to "on") compile, run, and plot out results. If one wants to rerun the model with a different time step for example, simply change the "**delt**" parameter (in the GENERAL RUNNING OPTIONS section) to a different value and set the "compile" option to "off" and relaunch the scm_job script.

The model will run for as long as it is set for a specific input data, see table 3.1. It is possible to change this time, but that is not recommended on the user level. Parameter "**mdays**" in armsubcas.cdk sets the number of days to run the program for. To have the run execute in time more precise than number of the days, "**kfinal**" and even some other parameters can be changed. But, changing these parameters in a wrong way will have the model run incorrectly. Therefore, they should not be changed for a regular user level run. More information on the parameters is in section 4.4.

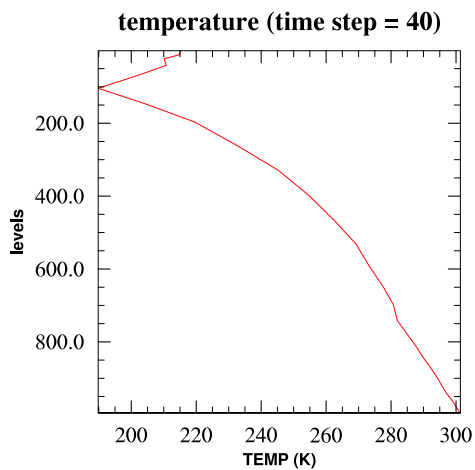
3.4 Run Examples

Figure 3.4 shows a few plots obtained using the ubc scm. Plotting routines (see Section 7) were used to get the graphs from the NetCDF file. The temperature file thts.out.nc was used for this example. (see Section 6.2)

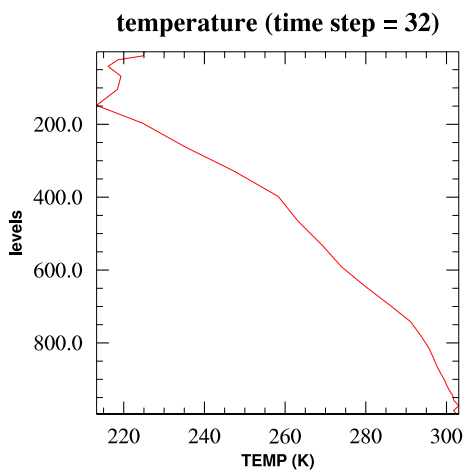
Model was run with different forcing file subcases (see 3.1) and each plotted for the largest time step value. The number of total time steps varies for each output.



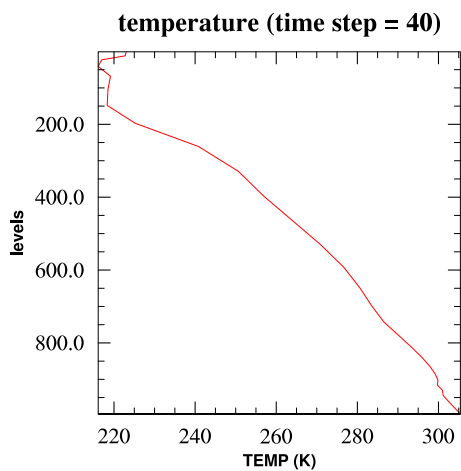
Temperature thts.out.nc
ARM_C1 X time=136



Temperature thts.out.nc
TOGA A time=40



Temperature thts.out.nc
ARM_C3 A time=32



Temperature thts.out.nc
ARM_C3 B time=40

4 Implementation

Changes in UBC SCM

CCCma SCM originally used the the CCCma's CCCRN binary data format for its output files. For the model to run at UBC without the use of the CCCRN format, the original program was adopted to use NetCDF binary file format for its output files. The drivers for the input/output files for the original CCCRN format were written in Fortran 77. These drivers were rewritten for NetCDF files in both, C++ and Fortran 90.

Programing languages

Most of the code was written in Fortran 77, but it is all compiled with a Fortran 90 compiler. The compiler depends on the system where the model is run. Plotting requires Python and NetCDF installed.

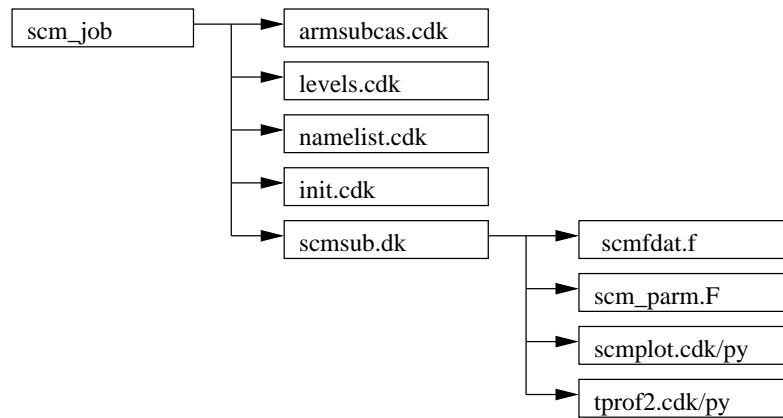
Flow Charts

Next couple of sections include all the **model Flow Charts** .

4.1 Scripts

The model is run using unix scripts. **scm_job** is the script that is run by the user (see Running The SCM Section 3.1). This script calls all the other scripts that initialize and set up parameters (see Parameters Section 4.4). The last script called, **scmsub.dk**, invokes the Preprocessor, main Program and two plotting scripts.

Figure 4.1 shows the sequence that these scripts are called in.



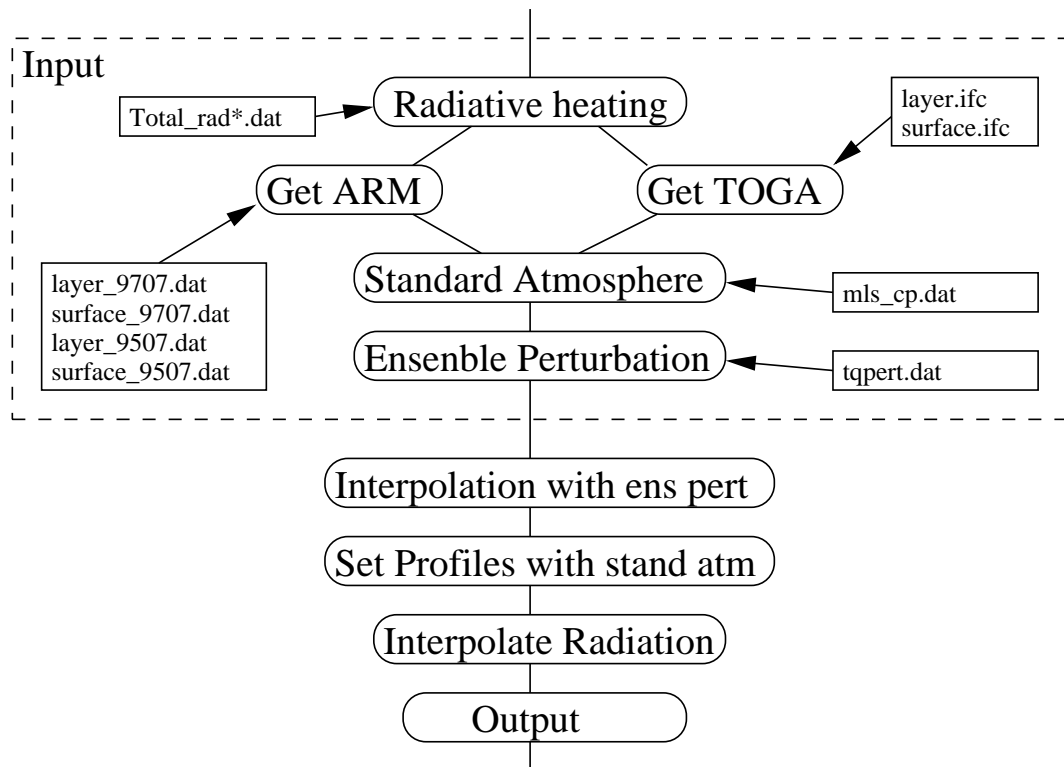
4.2 Preprocessor

Preprocessor `scmfdat.F` is invoked before the main program `scm_parm.F` (Figure 4.1). It reads in the raw text input data files and writes them into files in NetCDF data format. These files are then read by the main program.

In other words, preprocessor prepares the input data that the main program will work on. There is more information on these files in NetCDF Output Files Section 6.2.2

The input files are specified when the program is run (see Running the SCM Section 3.1).

Figure 4.2 is the flow chart for `scmfdat.F`. There is more information on what goes on in here in the Preprocessor Files Section 6.1.2.



4.3 SCM Algorithm

The code that runs the model is in the file `scm_param.f` (check Script Flow Figure 4.1).

Figure 4.3 shows the **model flow** .

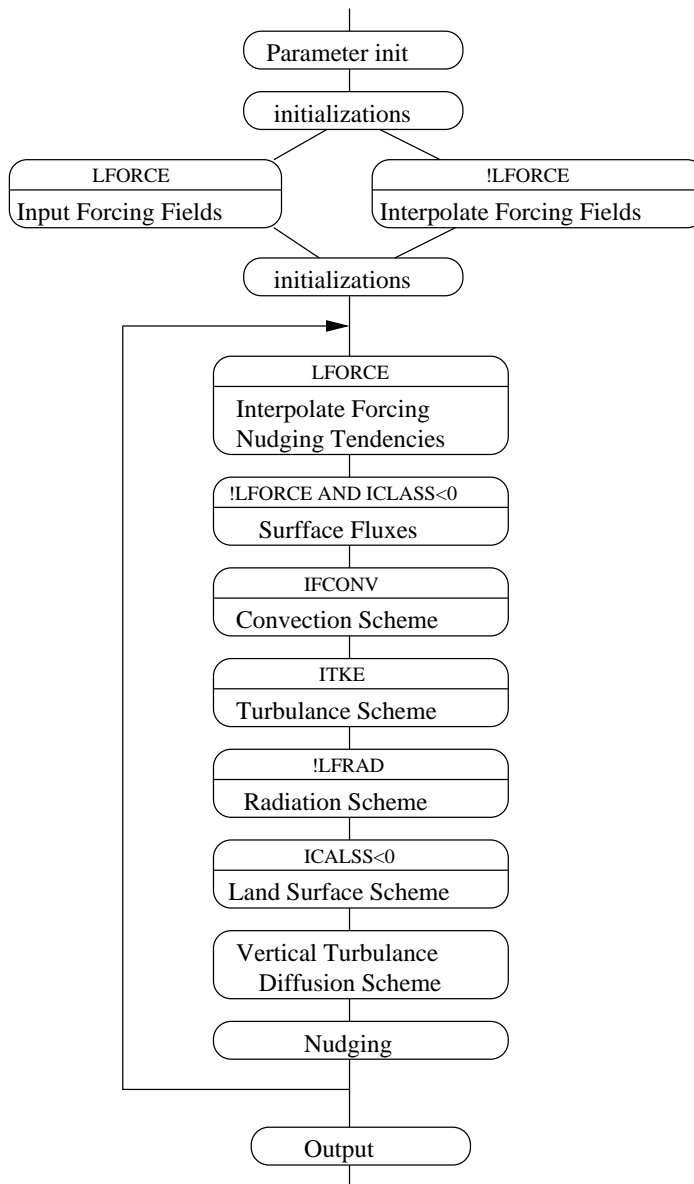
It starts of with Parameter initializations (`PARAM.DATA`) followed by other initialization for switches (`SET_SWITHCHES`), constants (`SPWCON7_S`), output files (`OPENTUBC`), pointers (`POINTP9`) and levels (`VERT`).

Then, if forcing is on, the forcing files will be read in (`GETS`) otherwise interpolation is done (`INTERPOL2`, `INITPRO`).

Further, there is more initialization for class (`HYDLAB`), arrays and Schemes (`TURBINIT`). After this, the model goes into a time loop where forcing, nudging and different schemes are done. The loop value is set to 75 in the fortran code.

The main loop involves four different Schemes (Convection, Turbulence, Radiation and Land surface Scheme). In each loop, forcing interpolation and nudging is done. Both dry and deep Convection Schemes make the Convection Scheme.

The only process not currently modelled is vertical momentum transfers due to gravity wave drag.



The following is a summary of all the schemes that are in the model right now.

4.3.1 Dry Convection Scheme

CONVEC10

Performs dry convective adjustments and computes both large and small scale precipitation.

also calculates relative humidity.

cond: *IFCONV* > 0

I/O : PCP, QH, TCV, TH, XROW

I : DEL, PRESSG, QSCRN, SHJ, SHTJ, SHXKJ,

4.3.2 Deep Convection Scheme

CONV6 (CONV4)

Performs deep convective adjustment based on mass-flux closure algorithm.

cond: *IFCONV* > 0

I/O : PCP, QH, T, U, UTG, V, VTG, XH

I : DEL, ICD, PBLT, PRESSG, SHBJ, SHJ, SHTJ, W

4.3.3 Turbulence Scheme

TKEDRIVER

cond: *ITKE* > 1

I/O :

I :

CLDOPT

cond: *ITKE* > 1

I/O : CMTX, TAC, CG

I : SHJ, SHTJ, TF, T, H, PSFC, PBLT, TCV, GC

CLOUDS9

cond: *ITKE* > 1

I/O : CMTX, TAC, CG, OMEGA, CLD, RH, CLW, CIC, CLWT, CICT, CLDT

I : SHJ, SHTJ, TF, T, H, PSFC, PBLT, TCV, GC

4.3.4 Radiation Scheme

OZON8_S (OZON8O, OZON8)

cond: !*LFRAD*

I/O : OZ

I : OZZXJ, PRESSG, SHTJ, JL

RADNEW7 (RADNEW5)

Calculate incoming visible and near ir at surface, corrected for local solar zenith angle at current time step. Initialize the cumulative arrays and define conversion constant. Process all levels and accumulate sigma vertical integrals. Convert to w/m2 and accumulate. Save heating rate time series. Compute precipitation rate from pret vector which represents precipitation depth in metres. Factor of 1000 is conversion factor from metres to mm (i.e., kg/m2).

cond: !*LFRAD*

I/O :

I :

4.3.5 Land Surface Scheme

CLASS13 (CLASS11)

Perform land surface processes calculations.

cond: ICLASSj0

I/O : HFSROL, QFSROL, TFX, QFX, ROFROL, QG, CDH, CDM, SFCT, SFCQ, SFCU, SFCV, ALSWROL, ALLWROL, EF, WF, WL, GT, SNO, ALBS, RHOS, TSNO, TT, TCAN, WCAN, SCAN, CMAI, SIC, SICN, ZSNO, GTROL, SICROL, SICNROL, GTROM, SICROM, SICNROM, RES, CBGO, BEGROL, BEG, BWGROL, BWG, MASK, DSIC, FS-

GROL, FC, FCS, SKY, SKYS, FSGV, FSGG, FLGV, FLGG, HFSC, HFSG, HMFC, HMFN, HTCS, HTCC, PCFC, PCLC, PCPN, PCPG, QFG, QFN, QFCL, QFCF, FSGS, FLGS, HFSS, ROFC, ROFN, ROFO, WTRC, WTRS, WTRG, HEVC, HEVS, HEVG, TBAR, THLQ, THIC, HTC, QFC, HMFG

I : FCAN, ALVC, ALIC, LNZO, LAMX, LAMN, CMAS, ROOT, SAND, CLAY, ORGM THL, QL, UL, VL, SGJL, SHJL, DSHJL, DLON, TS, QS, PRE, ENV, FSFROL, FDLROL, FSDROL, FSVH, FSIH, CSZ, PRESSG, GC, GTA, DR, ALSW, ALLW, DPTH, DRN, TBAS, FSNO, SNOF, WLAS, WFAS, WLA, WFA, QEVP, SNOM, RADJ

4.3.6 Vertical Turbulent Diffusion Scheme

VRTDF10

Calculate the tendencies of U,V,TH,Q,X due to vertical diffusion.

cond: TRUE

I/O : UTG, VTG, U, V, TH, TF, Q, TSG, TSGB, XROW, UTENDGW, VTENDGW, SGJ, SGBJ SHJ, SHBJ, SHTJ, SHTXKJ, SHXKJ, DSGJ, DSHJ, QFS, HFS, UFS, VFS, UFS-ROL, VFSROL CDH, CDM, PBLT, GC, GT, TS, QG, QFX, TFX, PRESSG, XFS, XFX, XSFX, RKXMIN, RKX

I :

4.4 Parameters

This section gives information on parameters more on the implementation level.

All the sizes of the variables used for interpolation as well as some other parameters are set in scripts `scmsub.dk` and `armsubcas.cdk` and written into files `sizes_fd` and `sizes`. The file `scmfdata.F` uses the parameters in “`sizes_fd` and” `scm_parm.F` uses the ones in “`sizes`”. Input data comes as three dimensional arrays (variable, levels, time) and all these sizes are written into these files.

The following references give a detailed information on most of the parameters used by the model. The preprocessor obtains most of the variable dimensions from the `sizes_fd` file.

<http://roc.eos.ubc.ca/users/ntucakov/ubscsm/guide/Readme-params>

The following is a summary of some examples for these parameter values. Parameter `sfdtout`, for example, varies for different subcases of ARM data.

<http://roc.eos.ubc.ca/users/ntucakov/ubscsm/guide/Readme-params-examples>

4.5 Variables

The following links are the files that contain information on variables that occur in the preprocessor `scmfdat.F` and the main program `scm_parm.F`.

The sizes of most of these variables depend on the values of parameters, section 4.4.

<http://roc.eos.ubc.ca/users/ntucakov/ubcscm/guide/Readme-varinfoSCMFDAT>

<http://roc.eos.ubc.ca/users/ntucakov/ubcscm/guide/Readme-varinfoSCM>

5 Software Overview

5.1 Software Components

This section gives an overview of all the files used in SCM, whether running or not.

5.1.1 All the components

This is a summary of all the files that are included in the ubcscm. It's given so that these files can be distinct from all the other files generated during the run, see section 6.2.

Creating the UBC SCM CVS

The following shows all the files that were copied from cccmascm to ubcscm to make the initial CVS version. To retain all the old RCS log messages, they were put into the file oldscmlogs.

```
>>>>
cd ~/projects/cccmascm/model
cvs log *.f *.F *.f90 *.cdk *.dk scm_job makefile *.inc *.DAT > oldscmlogs
>>>>
```

```
>>>>
cd ~/projects/cccmascm/model
cp *.f ~/projects/ubcscm/model
cp *.F ~/projects/ubcscm/model
cp *.f90 ~/projects/ubcscm/model
cp *.cdk ~/projects/ubcscm/model
cp *.dk ~/projects/ubcscm/model
cp scm_job ~/projects/ubcscm/model/
cp parmsub ~/projects/ubcscm/model/
cp makefile ~/projects/ubcscm/model/
cp makescmfdat ~/projects/ubcscm/model/
cp *.inc ~/projects/ubcscm/model/
cp varnames.nc ~/projects/ubcscm/model/
cp *.dat ~/projects/ubcscm/model/
cp *.DAT ~/projects/ubcscm/model/
```

```
cp *.ifa ~/projects/ubcscm/model/  
cp ANSCM ~/projects/ubcscm/model/  
cp Readme* ~/projects/ubcscm/model/  
cp oldscmlogs ~/projects/ubcscm/model/  
mkdir ~/projects/ubcscm/output  
>>>>
```

Note: ccc RCS didn't include some input data (fortin), although they were in the model directory, but here they are included in CVS.

5.1.2 Function Call tree

A nice tree structure of all the function calls within the the main scm program can be generated using a simple python script fdoc. The following page has more information on this. <http://roc.eos.ubc.ca/users/ntucakov/ubcscm/fdocinfo/index.html>

5.1.3 Extra components

All the Fortran files(function calls) that are not being used(called) with scm are listed below.

aeros2_data.f, angles.f, bascal.f, buoyan4.f, class11.f, class11_old.f, cldflx.f, cldprp4.f, closure4.f, colmod.f, conv4.f, coorddb.f, crvplot_col.f, cvmgt.f, datwbm2_data.f, datwbm3.f, decomp.f, deltae.f, deltae.s.f, forc_val.f, forcing.f, get_forc.f, get_nudg.f, hovplot.f, hovplot6.f, hovplot7.f, inithyd.f, initv.f, lenpak.f, levcal2.f, load_switches.f, longwv2.f, lwgrid.f, lwintg.f, lwpart.f, lwplnk.f, lwprof2.f, lwprof3.f, main.f, main_data.f, output.f, ozon8.f, ozon8o.f, pada2.f, path4.f, ploteph.f, ploteph_ARM.f, ploteph_alex.f, profav.f, profav2.f, profav_alex.f, profav_alex2.f, profil.f, profplot.f, putggb2.f, q1q24.f, radnew5.f, sdet2.f, shortw5.f, sigxk2.f, spwcon7.f, sweldfx.f, swlink2.f, test.f, tgcal2.f, tranlw.f, trfctn.f, trfctn2.f, trfctn.s.f, trtauo3.f, tsplot.f, tstream.f, tzplot.f, xplot.f, xplotrh.f, zmconv4.f, zxplot.f, micronlni.f90, lnblnk.f

Some are called only from scmfdat.F (lvdcode.f, writlev.f, coordab.f, upgr.f)

There are also calls within the model that have no reference. (gcmprep.f, inter.f, getarg.f,

system.f, getfld2.f, pkgr.f, getggb.f, initcld.f, molgth.f) Some calls are from ccnc (openubc.f, gets.f, puts.f, closeubc.f), some are temporary files(scm1.f).

5.1.4 Changes Made in SCM

After porting the SCM from ccma to ubc, there was a number of changes made to get i up and running. All the changes made are accompanied with a comment that includes “!nt”. An easy way to list all the changes in the program is to type “> *grep !nt **” in your ubcscm/model directory.

5.2 NetCDF Files

5.2.1 Installing NetCDF

A compressed tar file can be downloaded from:

<http://www.unidata.ucar.edu/packages/netcdf/index.htm>

uncompress with:

```
> gunzip netcdf.tar.Z
> tar xvf netcdf.tar
```

intallation instructions: [../netcdf-3.5.0/src/INSTALL.html](http://www.unidata.ucar.edu/packages/netcdf-3.5.0/src/INSTALL.html) or

<http://www.unidata.ucar.edu/packages/netcdf/INSTALL.html>

The following are the configuration parameters set for `ubc` and `ccc`. The examples are given for `bash` and `csh` shells.

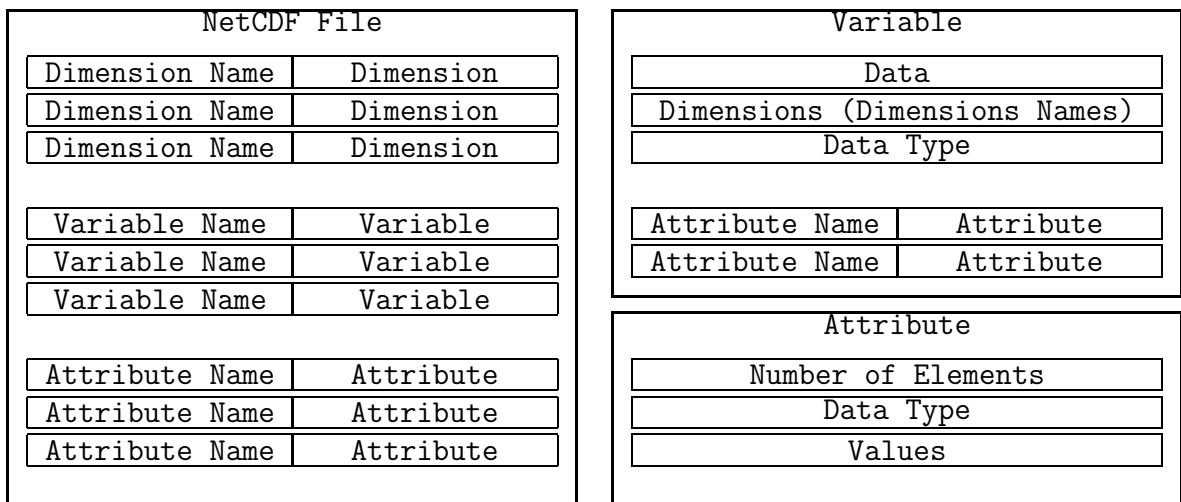
<code>ubc:</code>		<code>ccc:</code>	
<code>export</code>	<code>CC=cc</code>	<code>setenv</code>	<code>CC cc_r</code>
<code>export</code>	<code>CPPFLAGS=-DNDEBUG</code>	<code>setenv</code>	<code>CPPFLAGS -DNDEBUG</code>
<code>export</code>	<code>CFLAGS=-O</code>	<code>setenv</code>	<code>CFLAGS -O</code>
<code>export</code>	<code>FC=pgf77</code>	<code>setenv</code>	<code>FC xlf</code>
<code>export</code>	<code>F90=pgf90</code>	<code>setenv</code>	<code>F90 xlf90</code>
<code>export</code>	<code>FFLAGS=-O</code>	<code>setenv</code>	<code>FFLAGS -O</code>
<code>export</code>	<code>CXX=g++</code>	<code>setenv</code>	<code>CXX x1C_r</code>
<code>export</code>	<code>CXXFLAGS=-O</code>	<code>setenv</code>	<code>CXXFLAGS -O</code>

There was a problem in finding `g++` compiler but the following installation lines got us what we needed (`libnetcdf.a` and `ncdump`, `../lib/netcdf/lib/libnetcdf.a` and `../lib/netcdf/bin/ncdump`)

```
> cd intall/netcdf-3.5.0/src
> ./configure --prefix=~/.lib/netcdf/
> make clean
> make
> make install
```

5.2.2 About NetCDF

SCM program makes extensive use of NetCDF (Network Common Data Form), a file format maintained by Unidata that stores multidimensional arrays. Figure ?? illustrates the basic components of a NetCDF file. The file contains any number of dimensions, variables and attributes. A dimension contains only a single number that represents a length. A variable contains the data for a single multi-dimensional array along with its dimensions, data type, and any number of attributes. The dimensions associated with each variable are a subset of the dimensions contained in the file. Attributes, which can be associated with either a file or a variable, store either single numbers, strings, or one-dimensional arrays.



The contents of NetCDF files can be viewed using the `ncdump` program. For example, the command in Figure 5.2.2 uses the `-h` option to output the header information for a NetCDF file named `uq30_0_1.nc`.

```

> ncdump -h uq30_0_1.nc
netcdf uq30_0_1 {
dimensions:
    NZ = 34 ;
    NY = 120 ;
    NX = 160 ;
    N_ANGLES = 7 ;
    NZ_2 = 36 ;
variables:
    float FLUX_SUM(NZ_2, NY, NX) ;
    float FLUX_SQU(NZ_2, NY, NX) ;
    float RADIANCE_SUM(N_ANGLES, NY, NX) ;
    float RADIANCE_SQU(N_ANGLES, NY, NX) ;

// global attributes:
    :HISTORY = "\n",
    "2001 Dec 11 (12:03:53) Model saved\n",
    "2001 Dec 13 (20:43:09) Round 1 saved (nprocs=20,
        f77=0:29:03.59, eff=99.44%)" ;
    :MODEL_FILE = "uq30_0.nc" ;
    :ISEED = -100020 ;
    :ROUNDS_COMPLETE = 1 ;
    :TOTAL_PHOTONS = 192000000. ;
}

```

The `-v` option outputs data for specified variables. The command `ncdump -v RADIANCE_SUM uq30_0_1.nc`, for example, prints all 134,400 elements (`N_ANGLES x NY x NX`) of the `RADIANCE_SUM` array. The simple command `ncdump uq30_0_1.nc` with no options prints the data for all variables.

The program `ncgen` creates NetCDF files from text files formatted similarly to the output of `ncdump`.

5.2.3 UBC SCM NetCDF Data Format

Figure ?? gives an example of a header for a typical NetCDF data format used in SCM.

```
> ncdump -h thts.out.nc
netcdf thts.out {
dimensions:
    xdim = UNLIMITED ; // (32 currently)
    ydim = 137 ;
variables:
    double data(xdim, ydim) ;
        data:name = "TEMP" ;
    int levels(xdim) ;

// global attributes:
        :description = "temperature" ;
        :units = "(K)" ;
        :filename = "thts.out" ;
}
```

Dimensions

It has two dimensions, **ydim** being the number of time steps and **xdim** the number of levels.

Variables

There is two variables. Variable **data** holds its values in a two dimensional array (time vs levels).

Second variable **levels** holds the level values. The form of the levels (LH) is defined as follows:

```
SH = COORDINATE OR PRESSURE IN MB DIVIDED BY 1000.
    Read in vert.f, defined in namelist.cdk
LH = CODED LABEL FOR LEVEL IN THE FOLLOWING FORMS:
    Defined in siglab2.f/lvcode.f
1) -XAAA = A.AA E-X      IF SH < .010
2)  AAAA = AAAA         IF SH >= .010
```

The “xdim” being **unlimited** reflects the way the model was implemented. Data is written for each level, one at a time, and having it so is convenient. The current xdim value is the total number of levels and will not change for a specific run.

Attributes

The data values are described in the data variable’s attribute “name” and in the global attributes “units”, “filename” and “description” which are used in the plotting routines.

One Dimensional Array

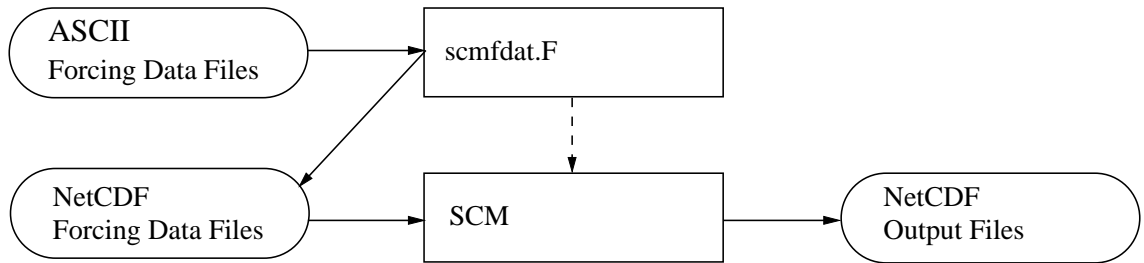
SCM uses this same NetCDF data format to record a one dimensional array. Everything is the same except the data in “levels” variable is not relevant and is set to 1. Figure ?? gives an example of this.

```
> ncdump -h capets.out.nc
netcdf capets.out {
dimensions:
    xdim = UNLIMITED ; // (1 currently)
    ydim = 137 ;
variables:
    double data(xdim, ydim) ;
        data:name = "CAPE" ;
    int levels(xdim) ;

// global attributes:
    :description = "cape" ;
    :units = "c" ;
    :filename = "capets.out" ;
}
```

6 Input/Output

The input and output files used by ubc scm are of NetCDF data format. More on this in sec ?? . Figure 6 shows where the **input and output** data files are used.



In case one wants to input a different data set, rather than the ones used currently by the model, there is information on how this can be done at the following site:
<http://roc.eos.ubc.ca/users/ntucakov/ubcscm/caseinsert/index.html>

6.1 Input Files

6.1.1 All Input Files

All the input files that appear in in the model can be split into two categories. The ones generated during the run and the ones that are not. The ones generated during the run are usually parameter definitions and some initializations. The ones that are not generated are data files such as forcing files and some standart input data. The scripts that run SCM initialize all the parameters and, convert the forcing files into a format readable by SCM.

Table 6.1.1 lists all the **scripts** used to run and initialize the model.

File name	used where	used with	comments
scm_job			Interactive submssion for scm
scm_parm.F	scm_job		=\$driver, Convert to scm1_parm.F in scmsub.dk
armsubcas.cdk	scm_job	script	Set forcing data params
levels.cdk	scm_job	script	Levels definition
namelist.cdk	scm_job	script	Create namelist parameter file
init.cdk	scm_job	script	Model initial conditions
scmsub.dk	scm_job		Model run
scmfdat.F	scmsub.dk	argument	parmsub the input forcing fields
(scmplot.cdk)	scmsub.dk	script	Time series plots
(tprof2.cdk)	scmsub.dk	script	Prifiles plots
coordab.f	scmfdat		
run_ensemble			

Table 6.1.1 lists all the **generated input files** using parameters.

File name	used where	used with	comments
scm1_parm.F scmfdat SCM	scmsub.dk scmsub.dk scmsub.dk scmsub.dk	argument argument ln execute	Parameter substitution for main driver Preprocess input forcing fields get a link since in the temp file
sizes sizes_fd Input_Cards Ftn_Unit_5 ../input/scminit ../output/ \$RUN_run/forc.*	scmsub.dk scmsub.dk scmsub.dk jclpnt.f jclpnt.f scmsub.dk scm_parm.F	argument argument argument unit 97 unit 5 ln unit 200:208	parmameters for scm1_parm.F parmameters for scmfdat.F for scmfdat overwritten by changed Input_Cards, Scratch, unit 98 Contains Cards from Input_Cards forcing files, NetCDF format
PARAM.DAT	scmsub.dk scm_parm	ln unit 100	parameters

Table 6.1.1 lists all the **data files** inputed by the model.

File name	used where	used with	comments
ma_iph15_an mls_cp.dat tqpert.dat	scm_job scmsub.dk scmfdat.F scmsub.dk scmfdat.F	ln unit 3 ln unit 7	=\$anfile, Not used in scm scm.dk any more standard atmosphere data formatted ensemble perturbation data formatted
layer_9507.dat layer_9707.dat layer.ifa	armsubcase scmsub scmsub scmfdat	assign ln argument unit 1	to \$ffile1 (ARM_C1) as \$ffile1 to scmfdat formatted, using jclpnt.f, read in GETARM \$ffile1 (ARM_C3) \$ffile1, read in GETTOGA
surface_9507.dat surface_9707.dat surface.ifa	armsubcase scmsub scmsub scmfdat	assign ln argument unit 2 TOGA	to \$ffile2 (ARM_C1) as \$ffile2 to scmfdat formatted, using jclpnt.f, read in GETARM \$ffile2 (ARM_C3) \$ffile2, read in GETTOGA
Total_radia*.dat	armsubcase scmsub scmfdat	assign ln unit 4	ARM_C3, to rad_ffile1 as rad_ffile1
ANSCM PRSCOZONE.DAT	scmfdat scmsub.dk ozonprep.f	unit NUGG=12 ln unit 1	class initialization get a link since in temp file read

In table 6.1.1 are some input files **not used** by the model any more.

File name	used where	used with	comments
FORCING.F	forcing.f	unit 5	this code is not used anymore, but unit 5 is in scmfdat!
fields.ifa	armsubcase	ln	TOGA
advect.ifa	armsubcase	ln	TOGA
eopo.ifa	armsubcase	ln	TOGA
misc.ifa	armsubcase	ln	TOGA

6.1.2 Preprocessor Files

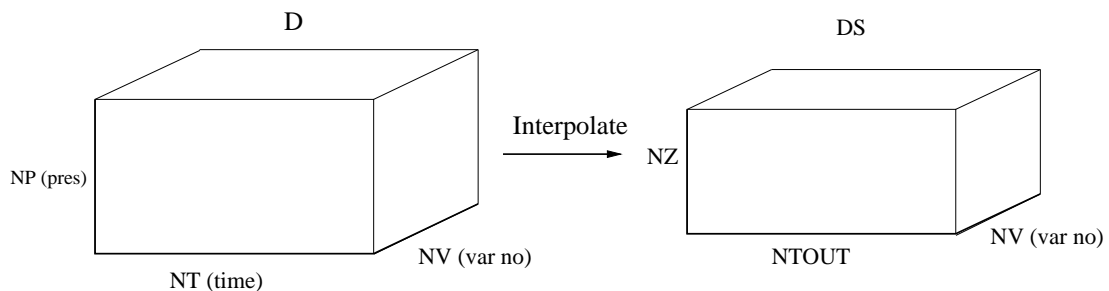
This section explains acquiring and interpolation of data in the preprocessor scmfdat.F. This is relevant when new data cases are to be inserted and used by the scm.

The format of input data is as follows. For each variable, data comes as one dimensional array for a series of levels. The size of these arrays is different for most input files. Each array is given for a specific point in time whose count varies from file to file as well. These sizes are saved in paramters explained in the Section 4.4.

in Seciton 4.2 give the folow chart for this preprocessor.

Here is in a bit of detal what happens in scmfdat.F file. First, the data is read in as plain text. Which data will be read is set when the program is run. After that, this data is interpolated to fit the dimensions used in the model. In other words, data is read with different dimensions for each case, but has to be set to match the model variables. The size for the level/preassure is always the same, although the value for time depends on the number of time steps set in the scm_job script.

Figure 6.1.2 shows an example of dimension change when **interpolating** .



The following is a summary of the dimension changes of the variables in scmfdat.

```

D      (NV, NP, NT) -> DN      (MAXVAR, NZ, NTOUT)
DS     (NVAR, NT)  -> DSN     (MAXVAR, NTOUT)
PRAD   (NPR, NTR) -> PNRAD   (NZ, NTR)
TPRIME (IPMAX, NPR) -> TNRPRIME (NZ)
QPRIME (IPMAX, NPR) -> QNRPRIME (NZ)

```

Table 6.1.2 lists all the preprocessor **input files** .

ARM_C1	ARM_C1	TOGA	standard atm	ensamble pert
layer_9507.dat	layer_9707.dat	layer.ifa	mls_cp.dat	tqpert.dat
surface_9507.dat	surface_9707.dat	surface.ifa		

Finally, this data is written out in NetCDF data format. See section 5.2.3 for more details on NetCDF files used by scm.

SCM model uses same dimensions that scmfdat interpolates variables to.

6.2 Output Files

6.2.1 All Output Files

Same as the input files, the output files can be split into the ones that are generated temporarily during the run and the ones that will “stay” after the run. Output files for each ensemble member are sent to the “./scm1.2/output/\$RUN_NAME_ensemble_member_run” directory. They are binary NetCDF files.

Table 6.2.1 lists all the files that are **generated temporarily** during the run.

File name	used where	used with	comments
scmfdat.f	scmsub	parmsub	Used to compile into scmfdat
sizes_fd	scm_job	cat	Used for parmsub in scmfdat.F
sizes	scm_job	cat	Used for parmsub in scl_parm.F
PARAM.DAT	namelist.cdk	cat	

Table 6.2.1 lists all the output files **used by the model** as well as the **final output files** .

File name	used where	used wiht	comments
scm1_parm.F	scmsub.dk	ln	Used to create the main driver, from scm_parm.F
scm1.F	scmsub.dk	parmsub,mv	Main driver
SCM	scmsub.dk	make/gnumake	
scmfdat	scmsub.dk	nogo	compiled scmfdat.f
../input/scminit	init.cdk	cat	
Input_Cards	scmsub.dk	cat	
Ftn_Unit_5	jclpnt.f	unit 5	Contains Cards from Input_Cards
../output/\$RUN_run/forc.*	scmfdat	units 11:16, 18::24,30:32	NetCDF format forcing files that will be used in SCM(scm_parm)
../output/\$RUN_run/*.out.nc	scmsub.dk	cp	save the files as output NetCDF format
	scm_parm	units 30:50 ...	
	scmsub.dk	mv	save the files as output
standard output		unit 6	

After a model run, there will be a number of new files created. Since the model runs in a temporary directory for some time, most of these files will be deleted after. Here is a summary of the files that will remain after the run and the ones that had already existed but were changed:

New output files: SCM, param_def, scmfdat, sizes, sizes_fd,

../output/\$RUN_NAME_ensemble_member_run/

Modified output files: scm1.F, scm1_parm.F

6.2.2 NetCDF output files

Table following references lists all the output files that are sent to `./scm1.2/output/$RUN_NAME_ense` directory. They are all in a NetCDF data format. See section 5.2.3 for more details on this. The first one is the list of all output files in the preprocessor and the second, all the output files from the main program.

<http://roc.eos.ubc.ca/users/ntucakov/ubscsm/guide/Readme-iolistSCMFDAT>

<http://roc.eos.ubc.ca/users/ntucakov/ubscsm/guide/Readme-iolistSCM>

7 Plotting

The following link describes how plotting is done with UBC SCM. It uses NCL scripts that read the NetCDF data format as its input data files.

To run plotting scripts, parameters "plotpr" and "plotts" in scm_job have to be set "on". Right now, only plotts will execute. It runs python scmplot_html.py script that generates a sample_plots.html web page containing some selected plots.

All the information and examles are provided.

<http://roc.eos.ubc.ca/users/ntucakov/ubcscm/ncplots/index.html>

8 More References

There is a number of other web sites that contain information related to most of the topics discussed in this report although most of them were already included within it. The following page contains a summary of references to all these links.

<http://roc.eos.ubc.ca/users/ntucakov/ubcscm/infopack/index.html>