MetCal and the SPA (version 1.7.1)

An extendible meteorological library and graphical interface

Ron McTaggart-Cowan
and the Mesoscale Research Group at McGill University

This manual describes how to use MetCal a graphically-driven meteorological diagnostics package and the expandable Shared Procedure Archive on which it depends (version 1.7.1).

Table of Contents

1 Overview 1

1.1 What is the SPA? 1
1.2 What is MetCal? 1
1.3 What data formats are acceptable? 1

2 The SPA 3

2.1 Installation of the SPA 3
2.2 Purpose of the SPA 4
2.3 Structure of the SPA 4
2.4 Extending the SPA 6

3 Metcal 7

3.1 Purpose of MetCal 7
3.2 File IO Page 7
3.3 Major Modes 7
3.3.1 Standard Outputs 7
3.3.2 Difference Calculation 8
3.3.3 Simple Math Calculation 8
3.3.4 Advection Calculation 8
3.3.5 Averaging 8
3.3.6 Filtering 8
3.3.7 Interpolation 9
3.3.8 Sounding 9
3.3.9 Cyclone Tracking 9
3.4 Standard Outputs Page 10
3.5 Modifying the Interface 10

4 Other Tools 11

4.1 Developer's Interface 11
4.2 PlotFST Plotting Utility 13
4.3 splitFST file management utility 14
4.4 fst2sig file conversion utility 14
4.5 trackinfo storm track management module 14

5 Customized Coding 16
5.1 Customized Program Templates 16
5.2 Setting the timeLoop 17
5.2.1 Full File Processing 17
5.2.2 Dates Processing 17
5.2.3 Range Processing 18
5.2.4 List Processing 18
5.3 Interfacing with Subprograms 19
5.4 Modifying Subprograms 20
5.5 Creating your own Subprograms 21
5.6 List of SPA Subprograms 22
5.7 Special Topics 26
5.7.1 Plotting with the SPA 26

Appendix A Copying Conditions 30
A.1 GNU Free Documentation License 30
A.1.1 ADDENDUM: How to use this License for your documents 36

Index 37

1 Overview

The Shared Procedure Archive (SPA) was originally developed at McGill University, starting in about 1999. The intention of the archive was to reduce create an easy-to-use diagnostics package for meteorological applications. It began as a set of portable FORTRAN77 functions and subroutines, and has since then grown into an evolving FORTRAN90 library with a Graphical User Interface (GUI, MetCal) and perl scripts which enable the developer's interface.

As noted in the licensing and copying agreement (see Appendix A [Copying Conditions], page 30) we encourage all users of this software to become developers and to contribute to the evolution of the SPA. Please share the library entries that you develop so that others may benefit from your efforts.

1.1 What is the SPA?

The SPA is a collection of FORTRAN90 modules, subroutines, and functions that create a working environment for standard meteorological calculations. The include data access routines (generally subroutines), setup and formatting routines (again, mostly subroutines), and calculation subprograms (generally generic functions). It is this latter set of routines that will be of primary interest to the user or developer who wishes to access or extend the SPA without introducing a new data format capability (which is, of course, much appreciated).

These subprograms are intended to abstract the user (and even the end-product developer) from the particulars of the data format supplied, and to make interfacing with existing routines as painless as possible. Inevitably, there will be small bugs in some of the calculations which will be caught and corrected over time — the end result will be a package which is as error-free as possible and meets the needs of users in the meteorological community.

1.2 What is MetCal?

MetCal (the "Meteorological Calculator") is the GUI for the SPA as described in Chapter 2 [The SPA], page 3. It is the interface that most new users of this package should become familiar with before they begin to work with the developer's interface (see Section 4.1 [Developers interface], page 11). The interface contains a notebook with three tabs, each of which is described in detail, See Chapter 3 [Metcal], page 7. The end-user simply has to define input and output files, along with the data type, and then request outputs from MetCal. The GUI, written in perl/Tk (a modular Tcl/Tk addon to basic perl), accesses a pre-compiled FORTRAN90 executable which handles the program flow at run-time. The result is a very simple interface for the user, who needs to know only basic information to access the computational routines of the SPA.

1.3 What data formats are acceptable?

Version 1.7.1 of the archive accepts gridded input data files in the following formats:

RPN Standard File (1989)

RPN Standard File (1998)

GEMPAK Data Format

As of version 1.6.0, the SPA also handles observational data stored in the following formats:

BURP Binary Universal Report Protocol

Rechèrche en Prévision Numerique (RPN) is a division of Environment Canada on whose in-house data storage format the original inception of the SPA was based. The data access routines stored in the RPN library (librmn.a) are backwards compatible between the 1989 and 1998 I/O subprograms, but the file structures themselves are significantly different, and must be distinguished for proper operation of the SPA. Note that the `ARMNLIB` environment variable must be correctly set during the installation of the SPA for the RPN data format access routines to be loaded.

The GEneral Meteorological PAcKage (GEMPAK) is an open-source distribution made available by the University Center for Atmospheric Research. The `GEMLIB` environment variable (usually set by the `Gemenviron` script distributed with the package) must be correctly set during the installation of the SPA for the GEMPAK data format access routines to be loaded.

As much as possible, all of the data to be used as input should be contained in a single input file, unless regridding is required.

New in version 1.6.0 is an application programming interface (API) for observational files. In the initial implementation, only BURP (Binary Universal Report Protocol) files are supported. These files are local to RPN, but are based on the more common BUFR format, so extension to that observational file type should be relatively simple.

2 The SPA

The SPA is a set of FORTRAN90 subprograms stored in a series of source directories beneath the SPA root. All interactions between the driving program and each of the subprograms are strictly controlled trough interface blocks stored in modules which are use-associated as needed in the code. This interfacing scheme allows for explicit error checking as well as making the subprogram interactions easily-understandable for the developer.

There are no common blocks or global variables in the SPA, although some common variables are declared within the lower-level modules responsible for file functions. These variables, however, are scoped only within the module and the subprogram by which it is associated. We are very insistent on the explicit passing of values between routines, since the data flow becomes rapidly untraceable once common blocks and global variables dominate in an extendible archive such as the SPA.

The subroutines contained in the SPA are primarily responsible for interactions with the datafiles and configuration scripts. Functions are generally used for calculation of diagnostics (the essence of the SPA) and are mostly generic, accepting either single value or array inputs for each argument, and producing single value or array output, respectively. Interfacing with array-valued functions is advisable, since they run many times faster than a nested single-valued function call.

2.1 Installation of the SPA

The SPA relies on several optional packages for full functionality, although the package can be built in a stand-alone format. The optional packages include ARMNLIB (for RPN Standard File processing), GEMPAK (for GEMPAK file processing), NetCDF (for NetCDF file processing), NCAR Graphics (for high-quality plotting), NCAR Spherepack (for highly-accurate computations on global domains), and Perl/Tk (for graphical interfaces). These packages, if required, should be installed in standard (system-dependent) locations before the SPA is configured.

The SPA uses GNU autotools to ensure platform-independence. In general, there are only a few steps involved in the installation process: configuration, building, and installing. The commands issued to perform these actions are (from the top level directory of the package):

```
./configure --prefix=/path/to/install
make all >& make.out
make install >&install.out
```

The `configure` script included with the distribution will usually do a good job of determining the setup of your machine. However, you may need to provide additional information (a list of options is available by running `configure --help`), especially if you are linking to a GEMPAK library built using different compilers from those used for the SPA build. Specifically, if FORTRAN77 compilers were used to build GEMPAK, then a set of `undefined reference` errors may occur when `configure` tries to link to the GEMPAK libraries. In order to address this problem, the `EXTRALIBS` variable is provided so that platform-specific conversion libraries can be appended to the library list for linking. For example, Sun Solaris platforms often require:

```
./configure --prefix=/path/to/install EXTRALIBS='-lF77 -lM77'
```

if native FORTRAN77 compilers have been used for compilation of the GEMPAK package. On Linux machines (depending on the compilers), it may look something more like:

```
./configure --prefix=/path/to/install EXTRALIBS='-lg2c'
```

if the G77 compiler was used for the GEMPAK build. Other FORTRAN77 compilers on Linux boxes may require links to the `libf2c.a` library instead (ie: `EXTRALIBS='-lf2c'`). On SGI machines, the problem is even more acute since the object file architecture is actually different for the native FORTRAN77 (`o32`) and FORTRAN90 (`n32`) compilers. Although we've had some luck using the `-n32/o32` compiler flags, it's best to build the GEMPAK library with FORTRAN90 from scratch on SGI workstations.

If GEMPAK is compiled with FORTRAN90 compilers, then *no* additional library definitions (`EXTRALIBS`) should be required. This is the best way to ensure the compatibily of the two packages; however, the FORTRAN90 build of the GEMPAK package can be a bit problematic.

For highly-accurate global calculations, the SPA supports an interface to NCAR SpherePack, a package which remaps and computes global values on a sphere rather than on the native grid of the dataset. The SpherePack package is well-tested and performs very well on global calculations within the SPA environment. Installation of SpherePack is, however, problematic since the source (available at www.scd.ucar.edu/css/software/spherepack is licenced under a highly-restrictive copyright that forbids the redistribution of the SpherePack source. It is therefore up to the installer of MetCal/SPA to download (for free) SpherePack and install it as `libsphere.a` somewhere on the default library path (`/usr/local/lib` is a favourite). If `libsphere.a` is installed in a non-standard directory, then this directory path can be given to configure at configuration-time as `SPHERELIB=/path/to/libsphere/directory` . A makefile for the SpherePack library can be found at www.atmos.albany.edu/facstaff/rmctc/spherepack . This makefile eases the installation process for the SpherePack package my allowing the user to enter `make all; make install` to install SpherePack to `/usr/local/lib` (must have write permission in this directory or install under a non-standard path such as `make all; make install PREFIX=$HOME` to install in `$HOME/lib`. Once `libsphere.a` has been properly installed, MetCal/SPA should be configured as outlined above for full functionality of the transparent SpherePack interface.

2.2 Purpose of the SPA

The SPA is intended to provide an easy means by which to create, store, and access meteorological diagnostics routines in a non data-format-specific sense. The layer of abstraction between the user (and the tools developer) and the data file allows for enhanced focus on the extension of the diagnostics database.

The premise for the SPA is that of an extendible community library of meteorological functions. Accordingly, contributions from as many sources as possible are encouraged. The SPA allows meteorological programmers to focus on the development and enhancement of the package, rather than on the development of lower-level interfacing software and basic functions.

2.3 Structure of the SPA

The SPA source is arranged in a set of subdirectories in this distribution. The naming of the source for the routines follows strict guidelines that are required for some of the automated build processes of the SPA. The routines are easily accessible through a series of developer's utilities described in See Section 4.1 [Developers interface], page 11, although most development work will occur in a local directory belonging to the developer (much like an RCS or CVS modification). The source subdirectories of the archive are:

'`subroutines`'
: This subdirectory is the repository for most of the generic initialization and setup utilities in the archive. The subroutines contained in this directory are usually accessed near the beginning of an SPA interface or when file I/O is required. All subroutines in the archive begin with the `sub_` prefix, followed by the name of the subroutine with a `.f90` extension.

'`functions`'
: The source in this directory primarily handles the calculations of the SPA. Functions are usually referenced either by control subroutines or by the developer in the "calculations" section of the code. All functions in the archive begin with the `func_` prefix, followed by the name of the function with a `.f90` extension.

'`modules`'
: The modules contained in this directory contain interface blocks for the control routines of the SPA. The interfaces are grouped by type (i.e. "mod_FST_File_Utilities.f90" holds all of the interfaces for RPN Standard File-specific code). All modules in the archive (with one exception noted in the next item) begin with the `mod_` prefix, followed by the name of the module with a `.f90` extension.

'`interfaces`'
: The interfaces for all of the calculation routines are contained in the interface file `Interface_MRG.f90`. This is the only source in the SPA whose name violates the rules described in the remainder of this table — and believe me I regret that every time I work with the code. A series of modules consisting of interface blocks are contained in the Mesoscale Research Group (MRG) primary interface, with subprogram access for each depends on broad groupings of functionality.

'`std`'
: The data file format-specific I/O subroutines are held below this subdirectory, with individual folders for each. Users with installed RPN, GEMPAK, and NCAR Graphics libraries (see Section 3.3.8 [Sounding], page 9) will have a fully-populated `std` subdirectory. Most of the files in this set of subdirectories are subroutines, although a few functions are also present for the GEMPAK interface.

'`opt`'
: This subdirectory has further subdirectories which allow the SPA to handle missing libraries without exceptions. For example, a user who has no need for GEMPAK data file structures, and who has no GEMPAK libraries installed, will see an `opt/gemlib` subdirectory filled with stub routines which warn the user about the library's unavailability and exit smoothly. For users with the RPN, GEMPAK, and NCAR Graphics libraries installed see Section 3.3.8 [Sounding], page 9 will see no `opt` directory in their distribution since all of the necessary routines are held below the `std` subdirectory.

‘`templates`’
: The template files in this subdirectory are intended to ease mundane coding tasks for developers by providing easy access to initiation and control routines in a usable fashion. The `template.f90` program is intended for standard development tasks, while the `template_gui.f90` source allows for GUI driven development.

‘`settings`’
: This directory holds an example set of configurations, again in an effort to ease the development process to the point where everyone will enjoy it and participate in it.

2.4 Extending the SPA

The easily-extendible nature of the SPA is one of the package's unique attributes. The SPA was originally designed by a devoted synoptician, and contained mostly large-scale atmospheric diagnostics. Other users have since requested (and often provided) very useful computational subprograms that have extended the utility of the package immensely. For example, you may want to calculate storm-relative helicity, a variable not currently supported in the SPA. The developer's interface provides a means by which you can build a helicity calculator with only a few read statements in the interfac3 (template) program. Our hope, of course, is that any computational routine that you create (or, any set of interface routines developed by the very brave) will be incorporated into the next release of the SPA.

A series of utilities are provided as the developer's interface as described in Section 4.1 [Developers interface], page 11. Following the steps described with the utilities will make development of the archive an almost enjoyable process.

3 Metcal

The METeorological CALculator (METCAL) is a GUI between the end user and the SPA. It consists of a set of three notebook pages on which the user sets File I/O characteristics, operation mode requirements, and desired outputs. Each of these pages is described in detail in the following sections.

3.1 Purpose of MetCal

MetCal is intended as an abstraction for the end user. As such, it is limited in flexibility compared to the Developer's Interface. However, it is easy to learn, and will be expanded as the SPA continues to grow. For off-the-shelf diagnostics, this utility can be extremely useful.

3.2 File IO Page

The File I/O page (the first page that the user will see following the copyright notice) allows the user to define the input and output files to be processed by MetCal. For both RPN file formats, the user also has the option to view any of the input or output files. The `primary input` file should reference the file containing all of the necessary input variable, if possible. For the "Standard Outputs" setting (see Section 3.3.1 [Standard outputs], page 8), the `secondary input` file can be used for supplemental input; however, all other Major Modes refer only to the `primary input` file. The `output` file defines the data file into which to store the output from MetCal. If this line is left blank, then the output is piped directly back to the input file. The `grid` file allows the user to remap fields to a desired output grid. All values read from the `primary input` and `secondary input` files are remapped to the domain contained in the `grid` file before the execution of any calculations. If the `grid` file is left blank, then no remapping of the input fields occurs (the default). The file referenced in the `grid` entry must contain at least one two-dimensional field of the grid onto which the data will be remapped.

All of the data times and levels present in the input file will be operated on by MetCal. For this reason, try as much as possible to isolate the data times that you require so that MetCal does not continue to execute on the entire set.

3.3 Major Modes

The Major Modes notepad of MetCal (selected using the tab at the top of the page) controls the program's data and control flows during execution. As described in the following sections, each Major Mode has a distinct functionality that may be of more or less use to different users. The default Major Mode is for Standard Outputs. Selection of any other modes produces a pop-up window for input from the user. All of the entries in the window must be completed before MetCal will execute.

3.3.1 Standard Outputs

The Standard Outputs mode of MetCal provides access to many of the calculation functions of the SPA. The Standard Outputs page allows users to select the calculations to be performed. Each calculation represents a routine (usually a function) contained in the archive. See Section 3.4 [Standard page], page 10, for details.

3.3.2 Difference Calculation

This simple field differencing algorithm can be used to compute the difference between a single variable at two different levels, or between two different variables at the same level. Of course, you could compute the difference between different variables at different levels, but you would really have to ask yourself why you are spending time doing that. The input and output identifiers must be the names of the variables contained in the `primary input` file on the page, See Section 3.2 [File io page], page 7.

3.3.3 Simple Math Calculation

This mode allows the user to perform simple (+, -, *, /) operations on the selected two or three dimensional field. The quantity in the `value` box may be either an integer or real number, but the computation takes place with a floating point number. This mode is especially useful for copying fields (multiply the desired field by 1) and for changing units.

3.3.4 Advection Calculation

The Advection Calculation mode provides access to the advection routine of the SPA. Any level of any field (under the `Field Information` heading) may be advected by the horizontal wind at any level. The vertical advection calculation is not implemented in version 1.7.1 of this distribution. Although the user may select from either the `primary` or `secondary input` data files by pressing the `Input Variables` button, this usage is discouraged and will likely be disabled in future releases, thus allowing only `primary input` file entries.

3.3.5 Averaging

The field averaging algorithm accessed by this Major Mode computes the average (either arithmetic or pressure-weighted) of a variable in a column whose upper and lower bounds are defined by the `Top Level` and `Bottom Level` entries. The input data for this mode must be given in the `primary input` file.

3.3.6 Filtering

The Filtering mode of MetCal runs a 25-point smoothing box over the entirety of the domain for the two or three dimensional variable specified in the `Identifier` entry. The shape and weights of the cells in the filter are shown graphically, and may be modified by the user to smooth more heavily or lightly, as required. There is no “magic number” up to

which the cell-totals must add, so feel free to modify the filter to fit your requirements. One of the drawbacks of the MetCal interface distributed with version 1.7.1 of this distribution is that the output identifier is set equal to the `Identifier`. This oversight will be corrected in future versions of the interface.

3.3.7 Interpolation

With horizontal interpolation handled automatically through the `grid` file, the Interpolation Major Mode is responsible for implementation of cubic-spline interpolation in the vertical. The pop-up window defaults to the mandatory pressure levels, but this is just an example of how to specify multiple level outputs. As many levels as required may be specified in a comma-separated list.

3.3.8 Sounding

The Sounding mode of MetCal requires NCAR Graphics libraries for plotting. If these libraries were not installed when the SPA was built on your machine, then you will not have access to the sounder. However, if the NCAR Graphics libraries were (and still are) available to MetCal, then the only entries required in the pop-up window are the x,y coordinates of the gridpoint at which to take the sounding (it will be taken at this point for every time in your input file, in keeping with MetCal tradition). The `Plot Title` entry is one of the few optional entries in the Major Modes. The soundings produced by this algorithm are in "metafiles" produced by NCAR Graphics, and can be viewed with the `idt` utility distributed with the NCAR Graphics package, accessible through the `view output file` button on the File IO page (see Section 3.2 [File io page], page 7).

3.3.9 Cyclone Tracking

The Cyclone Tracking Major Mode of MetCal is the most complex in terms of usage. Any two dimensional variable (in the `Identifier` field) can be operated on in the region of a tracked cyclone. Note that all cyclones are tracked using the height field (on pressure levels, default 1000~hPa in the `Level` entry) and that it must therefore be present in the `primary input` file, along with the field of interest. The `Maximum Cyclone Phase Speed` entry allows the user to track cyclones of different speeds. You will probably only run into a problem if this is set too low, but the default value of 30 m/s is already pretty high. The `Radius of Interest` defines the low-centre-relative region over which to operate on the field of interest. The default value for this entry is 1000 km.

The Cyclone Tracking Major Mode produces text output files of any or all of the field of interest's mean, standard deviation, minimum, or maximum near the low centre. These files are named with the `Text Output Root` prefix and `-mean`, `-sd`, `-min`, or `-max` suffixes. The output format is space-separated "`time value`" for forecast data, or "`date value`" for analysis data. These text files can be ingested directly into a plotting program such as `xmgrace` or (shudder) `Excel` to produce time-series of storm-following quantities.

Two execution modes are selectable by the user. The `automatic` start mode searches for the lowest height on the pressure level (i.e. the deepest low) in the initial field, and continues to track that feature. `Manual` start mode allows the user to input the x,y grid

coordinates of the cyclone to be tracked. Similarly, the `automatic` step mode allows the algorithm to track the cyclone automatically (or at least to attempt to). If the routine senses that it has lost the track, it will prompt the user for the x,y coordinate at a specific time in MetCal's parent shell. The `manual` step mode always asks the user for the location of the storm at every step.

Because of the unique interactive nature of this Major Mode, MetCal must be run in the foreground of the parent shell for correct functionality. We hope to rectify this problem with an interactive TopLevel window in future versions of the interface.

3.4 Standard Outputs Page

The `Output Fields` page is specific to the Standard Outputs Major Mode. Some of the more commonly used routines of the SPA are accessed by a series of check buttons. The `Input Variables` button allows the user to describe characteristics of the input fields contained in either the `primary input` or the `secondary input` data files. MetCal can, in general, accurately predict the fields that it requires to make the requested calculations. You should always check the input variables before execution. The `Check Variables` button allows the user to determine the output characteristics of the computed fields (essentially just the output variable identifier). Once you know these by heart for your usual file type, you probably will not need to use this button again. This page represents the beginnings of MetCal, and provides the most powerful functionality for the program.

3.5 Modifying the Interface

The GUI can be modified and updated by a developer who is familiar with perl/Tk. Access is provided via flags to the Developer's Interface (see Section 4.1 [Developers interface], page 11). In general, enhancements and upgrades should be distributed to the community and incorporated into the next release rather than being implemented in an ad-hoc fashion; however, if the development is just that good, then you have the flexibility to incorporate it between releases.

4 Other Tools

Several other tools are included with this distribution. Some of them relate exclusively to the RPN data file format, for which there is relatively little open-source software available. However, the Developer's Interface is central to the archive itself. Most of the tools distributed with version 1.7.1 of the SPA are described in the following sections.

4.1 Developer's Interface

The Developer's Interface is second only to MetCal as an avenue by which to access the SPA. This interface makes the job of developing routines almost enjoyable by relieving the developer from the tedious job of building an interface program. The basic steps involved in beginning a new development are:

1. Open a working directory and load template files and configurations;
2. Extract associated subprograms from the archive for verification of their functionality;
3. Determine dependencies and create a build environment (Makefile);
4. Write a new routine and interface through the template file; and,
5. Compile and test your new code.

Luckily, the SPA comes with a set of tools that make this process much simpler than it sounds. The important subprogram creation and template modification step (4) is outlined in detail in a subsequent section (see Chapter 5 [Customized Coding], page 16). In this section of the tools that makes up the Developer's Interface is described in detail in the standard order of their application. Each of these utilities is also supported by man pages. Please browse these pages for information not contained in this manual.

'`spaopen`' This utility completes Step 1 above, by creating a working directory and preloading it with the desired template and configuration files. `spaopen` accepts an optional flag and a single argument. The argument is the root name of the subdirectory, to which will be appended `_f90`. This is a hold-over from the FORTRAN77 days of the archive, but serves to emphasize the development directory nicely, so it stays. The recognized flags are: `-gui` which loads the GUI driven template, and a copy of the MetCal source code into the working directory; `-obs` which loads an observational-file handler template into the working directory; `-inv` which loads a PV inversion example and test program into the working directory; and, `-composite` which loads a compositing-ready template into the new directory (see Section 5.1 [Templates], page 16). Running `spaopen` without a flag (but still with the directory name) will result in a directory containing the standard interface template, and is the common usage of the utility.

'`spalist`' This utility allows the developer to look at a listing of the source code available in the SPA. The options for this tools are `subroutines`, `functions`, `interfaces`, and/or `modules` (`templates` may also work, but is of little practical use). These options of course refer to the subroutines of the archive. See Section 2.3 [Structure of the spa], page 5, for a complete description of the archive's structure. The ability to list the existing source is useful in two ways.

Firstly, it allows you to ensure that there is no repetition of subprogram names in your development, and secondly it provides information on the algorithms already coded in the SPA but not explicitly referenced (inasmuch as the subprogram names provide useful information about the functionality of the source).

'`spainfo`' This tool allows the developer to obtain detailed information about the functionality and interface of the specific routines listed as command line arguments. As of MetCal/SPA v1.3.9, the subprogram name can be given with or without suffixes (i.e. `.f90`) or prefixes (i.e. `sub_`, `func_` or `mod_`). The name of the subprogram is enough to have `spainfo` extract the appropriate header information. A file header, contained in the source, is printed to STDOUT and contains package, interface, version, and functionality information. There are two primary uses for `spainfo`. The first is to scan the archive for pre-existing routines which may be of interest to you. The second is in interfacing to pre-existing routines, since a full list of the arguments and their properties are displayed in the header. Note that `spainfo` produces a warning message if it finds a generic subprogram.

'`spaextract`'

Moving onto Step 2 above, the `spaextract` utility allows the user to obtain source from the correct archive subdirectory. Any number of routines can be obtained simultaneously. Again, the full file name must be provided so that `spaextract` knows where to look in the archive for the required source. Since MetCal/SPA v1.3.9, this restriction has been relaxed. The subprogram name is now all that is required, with the prefix (i.e. `sub_`, `func_` or `mod_`) and the suffix (i.e. `.f90`) now optional for additional extraction speed. Generic subprogram names are parsed before extraction and the user is alerted to the name of the file imported to the working directory.

'`comp90.d`'

Creating the build environment for your development project has never been easier. The `comp90.d` developer's interface tool allows the developer to painlessly develop a `Makefile` with a full dependency list and extract dependent files for recompilation in seconds (depending on the number of files to retrieve, of course). The only flag recognized by `comp90.d` is the `-opt` option, which compiles code in an optimized form. The compilation step can take quite a bit longer in this case, but optimization will yield returns at run time. `comp90.d` should be run any time that there is a modification to the source file listing of the directory so that all of its dependency lists are updated. Note that any SPA files that are automatically extracted by `comp90.d` are distribution cleanable. You should therefore run `make distclean` before re-running `comp90.d` in your working directory to clean out all of the automatically-retrieved code. Once the `Makefile` is built, it has the functionality of a standard GNU makefile.

The `comp90.d` utility requires some help in determining dependencies for function calls. If the source that you develop makes use of the functions `foo` and `foo2`, then your code should contain a line like:

```
! external :: foo,foo2
```

Although this is a comment line for the FORTRAN90 compiler, it will be correctly interpreted for the source's dependency list by `comp90.d`. The other anomaly associated with the `comp90.d` utility is the requirement of library loading, again displayed near the top of the template file:

```
! load FST       <- force comp90.d loading of ARMNLIB \
                                              (FST library)
! load GEMPAK    <- force comp90.d loading of GEMLIB \
                                              (GEMPAK library)
! load NCARG     <- force comp90.d loading of NCARG \
                                              (NCAR Graphics library)
```

This will ensure that the proper libraries are loaded by the linker to create the executable. (Of course, only the '`load XXXX`' is required - the comments are optional). Note that `NCARG` loading is not standard in the template file.

4.2 PlotFST Plotting Utility

The plotFST utility is a GUI which accesses the `sigma` plotting language developed by RPN. Clearly, this tool will only be useful for those using the RPN standard file data format. The interface is nearly entirely self-explanatory, and can be reformatted to either menus (default) or columns by selection in the `Display` menu. Both formats are functionally identical.

Several command line options are available to users, and can be listed with the `plotFST -help` command. The `-info` flag writes version and bug reporting info to STDOUT and exits; the `-file` argument should be followed by the input file for plotting (this can also be set easily within the application); the `-small` option runs the interface in a smaller window, thereby allowing users without large screen sizes to access all of the buttons.

Up to three variables can be plotted simultaneously (one for each "set"). The sets are activated by pressing the button at the top of the set's column. The initialized values are fairly common, so often there is not a lot of modification required before a usable product is obtained. Users familiar with the RPN standard file format will be used to dealing with the IP3 identifier, which can be used to specify fields beyond simple time and level stamps. The IP3 stamp is particularly useful for `plotFST` when running the utility on analysis data (in conjunction with `splitFST`).

The `Settings` menu allows the user to set map options and to create axis labels. However, it also aids in the quick production of plots. The `Locking` options refer to the propagation of time, level, and IP3 settings across sets. For example, if all three locks are engaged (as are, say, Sets 1 and 2) and the bar at the top of Set 1 is pressed, then the time, level, and IP3 settings of Set 1 are propagated to Set 2. If the hour lock is disengaged, then only the level and IP3 settings are propagated. This makes leveling and time changes much faster and practically bombproof.

Also under the `Settings` menu, the user is given the `Time Series` options of `Multiple Times` or `Multiple Files`. The time series output is engaged by entering a colon-separated list in the `Hour` entry box for the sets (i.e. `0:48:6`) of the form `start:end:step` in hours. This functionality is also available for the IP3 entry. The `Multiple Times` option produces

a single metafile (the output default output from `sigma`) with multiple times in it; the `Multiple Files` creates a separate file for each time, especially useful for GIF generation.

4.3 splitFST file management utility

The `splitFST` file formatting utility is usable only for RPN Standard data files. A complete man page for the tool is available online and contains a more detailed description of the capabilities of the tool than that presented here. The most useful application of the `splitFST` utility is in the plotting of analysis files using `plotFST`. The command `splitFST -dates` will provide each individual analysis time in a combined analysis file with a unique IP3 identifier. This allows `plotFST` to use its IP3 entry to isolate analysis times within the file. A full listing of the options accepted by `plotFST` can be obtained with the `-h` option.

4.4 fst2sig file conversion utility

The `fst2sig` file conversion utility is usable only for RPN Standard data files. A complete man page for the tool is available online and contains a more detailed description of the capabilities of the tool than that presented here. This converters is primarily useful in conjunction with the `plotFST` plotting utility also included with this package. It converts files containing locational parameters (`>>` and `^^` special vector fields) to self-navigating data fields which make use of the `ig1` through `ig4` parameters. This conversion is required for the plotting of cross-sections `plotFST` only.

Any file treated with this converter becomes an "old" formatted RPN datafile, so further access to the data will require the "RPN Standard 89" data file type setting in MetCal. However, since the latitude and longitude fields are not generated by the converter, extensive processing of the converted file with MetCal is not advisable.

4.5 trackinfo storm track management module

The `trackinfo` module is somewhat different from those listed above in that it is not a stand-alone executable script. Instead, this python module should be run interactively by the user from the python command line (for more information, see www.python.org). The most common use of this module is in the plotting of hurricane tracks from the NHC Best Track archive. The `trackinfo` module, can be used to produce the necessary input files for the `func_drawTracks.f90` function of the `NCAR_Utilities` SPA module, which in turn generates track maps and track overlays.

To invoke the python shell, type `python` at the prompt. Once the python prompt appears, you will need to load the `trackinfo` module by entering:

```
from trackinfo import *
```

(For python gurus, a full listing of the classes available in the `trackinfo` module can be written by using `import trackinfo` and `print trackinfo.__doc__` instead of the above). A list of the methods available to you from the trackinfo module can be obtained by entering the command:

```
print TrackList.__doc__
```

The first step is to initialize a new `TrackList` object by entering the command:

```
nhcBestTrack = TrackList('path/to/nhc/best/track','nhc')
```

The new object (`nhcBestTrack` - whatever you decide to call it - is now filled with all of the best track information from the NHC best track datafile (available at the NHC website www.nhc.noaa.gov). There are now a number of methods that you can apply to get useful information from the tracking dictionary. For example, you can create and list a subset of the 2001 hurricanes:

```
print TrackList.SubList.__doc__  #get info on the SubList interface
storms2001 = nhcBestTrack.SubList({'year':'2001'})
storms2001.PrintInfo({},'name:year:category')
```

This information can then be written directly to the format used by the SPA track plotting subprogram `func_drawTracks.f90` using a write method:

```
storms2001.WriteList('path/to/output/file')
```

This generates an SPA-compatible ASCII output file containing preformatted entries of all storms in the 2001 instance of the `TrackList` object. This file simply needs to be provided to the appropriate plotting function in the SPA.

Another potentially-useful feature of the `trackinfo` module is its ability to quickly identify storm locations. For example, to get a subset of the storms that have impacted the northwest coast of Florida, you can run something like:

```
FLstorms = nhcBestTrack.InBox([28,30],[83,84])
FLstorms.PrintInfo({},'name:year')
```

The attributes available to the user are:

`trackEntry` – full tracking information string.

`name` – name of the tropical cyclone.

`year` – year of occurence of the tropical cyclone.

`month` – month (numeric) of occurence of the tropical cyclone.

`day` – tropical cyclone start day.

`longMonth` – full month (string) of occurence of the tropical cyclone.

`shortMonth` – abbreviated month (string) of occurence of the tropical cyclone.

`intensity` – a list of intensity indicators for each reporting time.

`location` – a list of location indicators for each reporting time.

`category` – the category of maximum intensity for the storm.

Furthermore, a class variable `namedOnly` is defined for the `TrackList` class, which allows the user to control the reporting of information prior to the advent of NHC naming. The default pseudological value is 1 (off - no reporting of pre-naming storms); however, this is easily changed upon request:

```
TrackList.namedOnly = 0
```

Note that this value is immediately redefined for all instances of the `TrackList` class, and that none of the constructors need to be rerun.

5 Customized Coding

This chapter describes the Developer's Interface of the SPA in detail. It will be of most use to users who wish to interface with existing SPA subprograms (for simplified reading / writing and manipulation of datasets) or to extend the archive to include additional features. The tools provided with the MetCal to implement the Developer's Interface (see Section 4.1 [Developers interface], page 11) have already been described, and it is assumed in this chapter that the user is familiar with their implementation.

5.1 Customized Program Templates

Whenever a new project is begun with the `spaopen` (with flags as appropriate) command, a FORTRAN90 main program template is loaded into the newly-created directory. The template will have the same name as the project, but with a `.f90` extension. (Note that there is also a configuation file generated in the directory, named `settings.cfg` .) Although the template main program will compile and execute without modifications, any useful project will require the modification of the template to fit the specific requirements of the work.

Remember that the purpose of the main program is to interface with the subprograms of the library and of the project; this is *not* the place to implement calculations and new functionality since any such modifications will not be able to be transferred to the archive. As the archive changes, so too do the template programs, so a template that you created some time ago may no longer work with newer versions of the archive (although we try to maintain backwards compatiblity as much as possible). This is no problem if you've added a couple of read/write satements and a subprogram call to the template file, but will be particularly irritating if you have spent hours creating a monolithic main program out of the template. Obey good coding practices by using subroutines to abstract the functionality of your code and you will be much better off in the long run.

Each standard template file consists of nine parts:

1. Module loading (use association of SPA modules);
2. Program information and header;
3. Variable declarations;
4. Fetching of configuration information (from `settings.cfg`: call to `getParams`;
5. Initialization and opening of input/output files: call to `openGridFiles`;
6. Setup of the time-looping for the main program loop: call to `timeLoop`;
7. Determine grid information and allocate data arrays: calls to `getGridSize` and `getGridInfo`;
8. Creation of data descriptor, `dim`: call to `makeDim`;
9. Loop over times for read/manipulate/write: calls to `readGrid` and `writeGrid`.

As you can probably see from this list, much of the hard work involved in setting up for your calculations has already been done for you. Throughout the code (it would be a good idea to start a test project now so that you can follow along, if you haven't already) you will see MODIFY comments (only 5 in the current implemenation). These alert you

to areas in which you may have to make changes, specifically, items 1, 3, 6, 7 and 9 above. Most of these changes will be very minor (ie declaration and allocation of data arrays for your work).

For alternate templates, such as the composite and observational templates (accessed using the `-composite` and `-obs` flags to the `spaopen` call, respectively) the nature and number of modifications may vary. This is particularly true for the observational template where the calls, while conceptually similar, vary in syntax and type.

5.2 Setting the timeLoop

This section outlines some of the primary options available for timestepping through a dataset. Although there is a call to the `timeLoop` function already provided in the template file (see Section 5.1 [Templates], page 16), this will likely have to be modified to suit your needs. Although a complete description of template modification is reserved for subsequent sections (see Section 5.3 [Subprogram interfaces], page 19), this section will serve as a reference and introduction to the important time looping capabilities of the SPA. The `timeLoop` pointer function sets the *timer* variable in the main program segment, which contains a list of the times to be processed by the main program's primary time loop. The variations of the `timeLoop` setup allow for processing of full data files, individual dates and times, repeated intervals of dates and times, or date ranges. The primary functionalities of the `timeLoop` function are outlined here.

5.2.1 Full File Processing

Running in full file processing mode ensures that the grids for every valid input time in the primary given data file are handled in the main time loop. This mode is useful for model output data in which diagnostics at all times are desirable and is the original looping mode developed for the SPA. Its implementation is the simplest of any of the time processing modes.

```
timer => timeLoop('file',fileType,input(1))
```

This implementation sets tht *timer* to loop over all times in `input(1)`. Any of the input files can be used to define the main time loop.

5.2.2 Dates Processing

The dates processing mode of `timeLoop` creates a loop over repeated dates and times. This mode is particularly useful in developing climatologies. For example, if you need to compute the 6-hourly average of a field for June, July and August (JJA) from 1960 to 1990, then the dates processing mode will do the job for you. The implementation of the date processing mode can be somewhat daunting, but is simplified once you realize that each of the date/time components (year, month, day, hour, and fcst) can be described using either start/end values or a list of acceptable values. For example, JJA can be requested either as `startMonth=6,endMonth=8` or as `listMonth=(/6,7,8/)`. Clearly the style that you choose will depend on the length and sequence of the list. For the example outlined above, the call to `timeLoop` would look like this.

```
timer => timeLoop('dates',fileType,startYear=1960,endYear=1990, &
 listMonth=(/6,7,8/),startDay=1,endDay=31,startHour=0,endHour=23, &
 incHour=6,listFcst=(/0/))
```

Note that the end time does not need to be present in the dataset - if it is not, then valid time *up to* the end time will be treated. To loop over the 12-hourly values of January 2nd to 15th fields from 1980 to 1994, your call would look something like this.

```
timer => timeLoop('dates',fileType,startYear=1980,endYear=1993, &
 listMonth=(/1/),startDay=2,endDay=15,listHour=(/0,12/),listFcst=(/0/))
```

Suppose now that you want to treat the 12-hourly December, January and February (DJF) values for a field from 1980 to 1999. The catch here is that you actually want to use the December 1979 fields in the 1980 start year since you're doing a wintertime climatology. To do this, you must provide a list of input months, with negative values for months from the previous year. The previous December is -12, with the previous November as -11, and so on, as shown in this example.

```
timer => timeLoop('dates',fileType,startYear=1980,endYear=1999, &
 listMonth=(/-12,1,2/),startDay=1,endDay=31,listHour=(/0,12/))
```

5.2.3 Range Processing

The range processing mode of `timeLoop` creates a loop beginning on a given date and ending on a second given date. All defined dates and times between the two dates are processed. This can be useful for generation of a mean state over multiple days, months, or years. Note that the date/time components (year, month, day, hour, and fcst) are identical to those for the dates processing mode (see Section 5.2.2 [Dates processing], page 17) but that all should be defined only with start/end values since lists are nonsensical in this context. For example, if you wished to loop over all 6-hourly periods from 20 March 2001 to 14 July 2004 (for whatever reason), then you could implement the following call to `timeLoop`.

```
timer => timeLoop('range',fileType,startYear=2001,endYear=2004, &
 startMonth=3,endMonth=7,startDay=20,endDay=14,startHour=0, &
 endHour=23,incHour=6)
```

Note that the end time does not need to be present in the dataset - if it is not, then valid time *up to* the end time will be treated. Now suppose that you want to loop over all 12-hour times in January 2004.

```
timer => timeLoop('range',fileType,startYear=2004,startMonth=1, &
 startDay=1,endDay=31,startHour=0,endHour=23,incHour=12)
```

Note that the end values of each variable default to the start values if they are not provided. This shorthand makes it quicker to define short ranges as above.

5.2.4 List Processing

The list processing mode of the `timeLoop` handles a series of comletely independent input dates. This mode is particularly useful for compositing, where seemingly random dates are accessed and handled together. In this case, the dates and times are given as a vector of strings to the `timeLoop` function. The order of the string is: `yyyy/mm/dd/hh/fff` , where

`fff` is a three-digit forecast time (optional, with a default of `000` for analysis data. To loop over a series of dates, implement something similar to the following.

```
timer => timeLoop('list',fileType,dates=(/'1963/01/10/00', &
 '1972/02/18/12','1980/04/28/18','2001/02/11/06'/))
```

This will loop over analyses from the given dates/times. Note that the *first* input string is used to determine the maximum length of all subsequent date definitions, so if you wish to include forecast specifiers in the loop, then this must be done for at least the first date string. For example, if the first string were `1963/01/10/00` and the second string were `1972/02/18/12/006`, the second string would be clipped (without warning) to `1972/02/18/12`. The correct way to implement this looping would be to define the first date as `1963/01/10/00/000`, which would result in the expected handling of the second date string. Thus, when in doubt, err on the side of providing as much information as possible to the list.

5.3 Interfacing with Subprograms

An example may ease the process of learning to modify the template files and how to use the Developer's Interface to access the subprograms stored in the archive. Let's say that we want to compute the potential temperature for all times in our input file, but for some reason do not wish to use MetCal. Using the developer's interface, we would do the following.

1. Start a new project called `ptmp` by entering the command `spaopen ptmp`;
2. Enter the new directory using `cd ptmp_f90`;
3. Begin editing the `ptmp.f90` file using our favourite text editor (ie xemacs, vi, etc);
4. Look for a potential temperature calculator in the SPA using either this manual or `spalist` (we find it as `func_tttopt.f90`);
5. Examine the subprogram header information to determine which module we need to include in the main program to access the potential temperature function (we use `spainfo tttopt` and find that it is part of the `Thermodynamics` module);
6. We now modify the template file (section 1) to use-associate the thermodynamics module by adding `use Thermodynamics` near the first MODIFY comment in the template (`ptmp.f90`);
7. Now we scroll down to the next MODFIY comment and find that we need to declare additional data arrays (the `tt` array is defined by default as an example) - to allow us to store our newly-computed potential temperature, we'll add the `pt` 3D array by modifying the declaration line to look like:

   ```
   real, dimension(:,:,:), allocatable :: tt,pt
   ```

8. Continuing on to the next MODIFY statement, we find that the `timeLoop` call will act on every time in the file, which is what we want so no changes are needed (see Section 5.2 [Setting the timeLoop], page 17);
9. The next MODIFY comment reminds us to allocate our working grids - since we added the 3D `pt` array, we need to add a new line containing something like: `allocate(pt(ni,nj,nk))`;

10. Now we're ready to change the reads / writes and calls to suit our needs as indicated by the next MODIFY comment. First we check that we are reading all of the required data (we can check in again with `spainfo tttopt` if we've forgotten what needs to be provided). In this case, we're fine since all we need is temperature (the default) - make sure here that the name of the temperature variable (RPN Standard 'TT' by default) matches with the dataset. Under the `Calculations here...` comment, we add the call to the required function: `pt = tttopt(tt,dim)` and then modify the `writeGrid` call to output the potential temperature field. Something like: `call writeGrid(output(1),pt,'PT',timer(t,:),dim)` will work just fine;
11. That is all we need to do for the main program - now we just have to modify the well-documented `settings.cfg` file to provide the file format, and names of the input and output files;
12. Run `comp90.d` to generate the Makefile;
13. Run `make ptmp` to create the `ptmp` executable; and,
14. Run the `ptmp` executable to compute the potential temperature field.

Although this list may look daunting at the outset, such simple modifications can take as little as a minute once you become familiar with the SPA. The code that you have developed is now ready to handle any input file containing the defined temperature field in either RPN Standard or GEMPAK format without recompiling. Congratulations on your first development with the SPA.

5.4 Modifying Subprograms

Suppose now that you are not happy with the result of your efforts in the previous section. You wind up with a potential temperature field that defies hydrostacy and you want to know why. Is something wrong with the `func_tttopt.f90` subprogram in the SPA? That is, of course, always possible - so you want to have a greater level of control over the function.

Begin by following all of the steps in the interfacing section (see Section 5.3 [Subprogram interfaces], page 19) - which you presumably have already done to get to the problem. The following steps will allow you to gain complete control over the SPA's calculation of potential temperature.

1. Run `make distclean` to remove any dependency-related extractions that may have ended up in the project directory;
2. Enter `spaextract tttopt` to create a local copy of the subprogram's source in the project directory;
3. Use your favourite editor to hack the source in any way you see fit;
4. Run `comp90.d` to regenerate the Makefile with updated dependencies for the `func_tttopt.o` object; and,
5. Enter `make ptmp` to create the executable.

It is as easy as that to obtain, modify, and recompile sections of the SPA. Note that the SPA keeps track of internal dependencies and extracts dependent files into the project directory during step 4 above. These files should not be modified since the `make distclean`

command will remove them. Should you subsequently wish to modify one of the dependent files, make sure to repeat all of these steps (in order) so that you do not end up modifying a temporary file.

Changing the argument interface to SPA subprograms is a little more difficult, but certainly not impossible. For this, you will need to extract `Interface_MRG.f90` along with the subprogram of iterest. Once you have made your modifications to the subprogram argument list, find the related interface block in `Interface_MRG.f90` and modify it accordingly. At this point, running `comp90.d` will cause the extraction of *many* subprograms, but once they are compiled they will be of little consequence to the project.

5.5 Creating your own Subprograms

The next step in interacting with the SPA is to release yourself from the limitation of using only subprograms provided with the archive. For example, let's say that you've just come up with the latest and greatest atmospheric variable of all time and decided to call it *foo*. But of course the SPA doesn't know all about your brilliant insight, so you need to teach it how to compute foo. For simplicity, we'll assume that the calculation of foo requires only the temperature field, so that all of the basic modifications are identical to those outlined in the interfacing section (see Section 5.3 [Subprogram interfaces], page 19). (We can, of course, skip the step involving the use-association of the thermodynamics module if we wish since our new subprogram will exist locally.)

Instead of calling the `tttopt` function, we'll call the `foo` function for our calculation. This will look something like: `pt = foo(tt,dim)`. (Note that we're now storing our precious foo variable in the pt data array, but that will work fine in the end so we'll go with it for simplicity.) *Before* generating the Makefile using the `comp90.d` command, however, we need to create the `func_foo.f90` subprogram. The naming of this external function is important, since it is through the program name that the Makefile dependencies are generated. All functions must have the `func_` prefix, and all subroutines must have the `sub_` prefix. You should try to program primarily functions for the implementation of simple calculations.

To get started writing `func_foo.f90`, we can extract a simple function from the archive and imitate the format found within. For example, we could run `spaextract density` and then use `func_density.f90` as a "template" for our own subprogram. All coding of SPA subprograms must conform to FORTRAN90 standards - consult FORTRAN90 manuals for assistance in writing subprograms. If an interface is required for your newly-developed `func_foo.f90` (array-valued return, optional arguments, or just general safety), then insert it following the variable declarations in the main program before attempting to compile the project.

Now that you have `func_foo.f90` ready to go (and making sure that you've removed the local copy of `func_density.f90` that you used as a guide), enter `comp90.d` to generate the Makefile. If your nomeclature is correct, then you should be able so see a dependency in the Makefile: the main program will depend on the func_foo.o object file. Now you're ready to compile and run the project by entering `make exec` (or the project name) and runnin the resulting executable (once you've worked out any compile-time errors).

Once you've checked (and double checked, and had your buddy down the hall check) `func_foo.f90` and are convinced that all is copacetic, the most important step

of SPA development arrives. In order to allow all users access to your fantastic foo variable, you need to submit `func_foo.f90` to the SPA. You can do so by contacting the developers or using one of the MetCal mailing lists on the SourceForge servers (http://sourceforge.net/projects/metcal). Provide as much information and documentation as possible to ease the introduction of the comptation into the archive. Please do not omit this very important step since it is the only way that the SPA will continue to grow and be of use to the community.

5.6 List of SPA Subprograms

This section provides a list of the available high-level SPA subprograms. This section focuses on calculation and I/O subprograms, the most likely to be modified and accessed by developers. While this list is as up-to-date as possible, the `spalist` utility will provide better information on the subprograms present in your version of the package. No interfacing information is provided here since changes are difficult to document. The `spainfo` utility should be used to obtain interfacing information.

`func_absVort.f90`: compute absolute vorticity (Vorticity).

`func_adv.f90`: compute advection (Math).

`func_anomalyCorrelation.f90`: compute the anomaly correlation score over a given grid area (Math).

`func_avg.f90`: compute averages in the vertical (Math).

`func_areaAverage.f90`: compute the mean of a quantity over a specified horizontal area (Math).

`func_balance.f90`: compute streamfunction or geopotential using either geostrophic or gradient balance equations (Dynamics).

`func_bvFrequency.f90`: compute the Brundt-Vaisala frequency of the atmosphere (Thermodynamics).

`func_colInt.f90`: compute column integral of a quantity (Math).

`func_coriolisForce.f90`: compute the Coriolis force (Vorticity).

`func_couplingIndex.f90`: compute the coupling index (otherwise known as the coupling potential) (Thermodynamics).

`func_createColourMap.f90`: generate a new NCAR Graphics colour map (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_dateStamp.f90`: convert yy/mm/dd/hh times into internal date stamp and vice versa (Math). `func_dateStampString.f90`: convert internal date stamp into a set of strings (Math).

`func_deformation.f90`: compute deformation magnitude and dilitation axis orientation (Dynamics).

`func_density.f90`: compute density from the ideal gas law (Thermodynamics).

`func_der.f90`: compute the derivative of a field at a given point and in a given direction (Math).

`func_derSq.f90`: compute the second derivative of a field at a given point and in a given direction (Math).

`func_diff.f90`: compute the difference between two (possibly) different fields at two (possibly) different levels (Math).

`func_divergence.f90`: compute divergence (Dynamics).

`func_drawCentered.f90`: plot a storm centered field on a polar stereographic projection with the pole at the point of interest (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_drawFill.f90`: fill a specified region with a specified colour (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_drawLine.f90`: plot a line on a horizontal map (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_drawPinwheel.f90`: use a 2D wind field and a center location to draw a summary chart of hurricane intensity and symmetry (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_drawPlot.f90`: generate a plot of a two-dimensional field as an NCAR Graphics metafile (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_drawSounding.f90`: generate a skew-T type sounding for data provided (temperature, humidity and winds) (available only if NCAR Graphics is installed) (NCAR_Utilities)

`func_drawTracks.f90`: generate a plot of storm or hurricane tracks as an NCAR Graphics metafile (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_drawTrajectory.f90`: generate a plot of a parcel trajectory as an NCAR Graphics metafile (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_drawText.f90`: plot text on a map (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_drawVectors.f90`: generate a plot of a two-dimensional vector field as an NCAR Graphics metafile (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_dynTrop.f90`: compute a quantity on the the dynamic tropopause (Dynamics).

`func_ertelPV.f90`: compute Ertel potential vorticity (Vorticity).

`func_eqtott.f90`: convert equivalent potential temperature to dry bulb temperature while holding either specific humidity or relative humidity constant (Thermodynamics).

`func_estohr.f90`: convert dewpoint depression to relative humidity (Thermodynamics).

`func_estohu.f90`: convert dewpoint depression to specific humidity (Thermodynamics).

`func_frequencyFilter.f90`: run a high/low/band pass filter over a complete dataset (Math).

`func_getPower.f90`: get the order of a floating point number (Math).

`func_hrtoes.f90`: convert relative humidity to dewpoint depression (Thermodynamics).

`func_hrtohu.f90`: convert relative humidity to specific humidity (Thermodynamics).

`func_hutoes.f90`: convert specific humidity to dewpoint depression (Thermodynamics).

`func_hutohr.f90`: convert specific humidity to relative humidity (Thermodynamics).

`func_hydrostatic.f90`: compute hydrostatic layers or temperatures from the other component of the mass field (Thermodynamics).

`func_interpolate.f90`: interpolate from one grid to another (Math).

`func_isentropic.f90`: interpolate an user-defined variable onto a given isentropic surface (Dynamics).

`func_isentropicPV.f90`: computes isentropic Ertel potential vorticity on a set of isentropic surfaces (Vorticity).

`func_isGlobalGrid.f90`: determines whether the active grid is global or regional (Math).

`func_julianDay.f90`: compute the Julian Day of the data field (Math).

`func_Lap.f90`: compute the Laplacian of a field (Math).

`func_LapInvert.f90`: invert the Laplacian of a field (Math).

`func_lc.f90`: convert string to lower case (String).

`func_lpad.f90`: pad the left side of a string with a specific character (String).

`func_ltrim.f90`: remove whitespace from the left side of a string (String).

`func_mapScale.f90`: compute map scale factor (Math).

`func_moistureConvergence.f90`: compute moisture convergence (Thermodynamics).

`func_ncarEnd.f90`: manual shutdown of the NCAR Graphics plotting system (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_ncarInit.f90`: manual initialization of the NCAR Graphics plotting system - can be used to generate PostScript files instead of default metafiles (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_ncarMap.f90`: initialize a standard NCAR Graphics map (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_obsFileSize.f90`: obtain the number of station entries in an observational file (TopLevel).

`func_pathLength.f90`: compute the true distance between two points on the sphere (Math).

`func_penetrationDepth.f90`: Compute the Rossby penetration depth of the atmosphere (Dynamics).

`func_Performance.f90`: timer to clock performance of the SPA - note that the naming strategy in this subprogram is non-standard (Performance).

`func_potentialIntensity.f90`: compute the (maximum) potential intensity of the atmosphere from K. Emanuel's model (Thermodynamics).

`func_powerDissipation.f90`: compute the power dissipation of a storm modeled by a Rankine vortex. The computation of the power dissipation folows Emanuel (2005) [Nature] (Dynamics).

`func_pvmc.f90`: compute the moist component potential vorticity (Vorticity).

`func_resetColourMap`: resets the given (default) NCAR Graphics colour map (available only if NCAR Graphics is installed) (NCAR_Utilities).

`func_slpReduction.f90`: compute the sea level pressure from the mass field (Thermodynamics).

`func_staticStab.f90`: compute the static stability (Thermodynamics).

`func_streamFunction.f90`: compute the streamfunction (Dynamics).

`func_tcCentre.f90`: determine the centre of a TC (Dynamics). `func_timeLoop.f90`: determine time structure required by user (TopLevel)

`func_track.f90`: track a disturbance (Dynamics).

`func_tTest.f90`: compute the statistical significance of a field (Math).

`func_tttoeq.f90`: convert temperature to equivalent potential temperature (Thermodynamics).

`func_tttopt.f90`: convert temperature to potential temperature (Thermodynamics).

`func_uc.f90`: convert string to upper case (String).

`func_velocityPotential.f90`: compute the velocity potential (Dynamics).

`func_verticalIndex.f90`: determine the index of a given level (Math).

`func_vertInt.f90`: perform cubic vertical interpolation (Math).

`func_windcomp.f90`: compute wind x/y components (Dynamics).

`func_winddir.f90`: compute wind direction (Dynamics).

`func_windspd.f90`: compute windspeed (Dynamics).

`func_wptowz.f90`: compute vertical motion from omega (Dynamics).

`func_wztowp.f90`: compute omega from vertical motion (Dynamics).

`sub_apeBudget.f90`: compute the right-hand-side terms for the eddy available potential energy (APE) equation (Dynamics).

`sub_cape.f90`: compute CAPE and CAPE-based variables (Thermodynamics).

`sub_closeObsFiles.f90`: close a set of observational files (TopLevel).

`sub_composite.f90`: compute the running mean of a variable (Compositing).

`sub_EadyAtmosphere.f90`: compute a basic-state atmospheric structure based on the constraints of the Eady model (Dynamics). `sub_ekeBudget.f90`: compute the right-hand-side terms for the eddy kinetic energy (EKE) equation (Dynamics).

`sub_fillMissing.f90`: fill in missing values on pressure surfaces below the ground and above the model lid (Math).

`sub_filter.f90`: apply a 25-point smoothing filter (Math).

`sub_frontogenesis.f90`: compute Miller frontogenesis and component terms (Dynamics).

`sub_garbageCollection.f90`: perform garbage collection duties for some memory-intensive SPA compontents (TopLevel).

`sub_grptLocator.f90`: determine the gridpoint closest to the given lat/long values (Math).

`sub_irrotationalWind.f90`: compute the irrotational component of the wind (Dynamics).

`sub_moistureTransport.f90`: compute the moisture transport and component terms (Thermodynamics).

`sub_ncarCentered.f90`: compute a storm centered grid with the point of interest at the pole (available only if NCAR Graphics is installed) (NCAR_Utilities).

`sub_ncarPanels.f90`: compute positional information for a specified number of panels on a page (available only if NCAR Graphics is installed) (NCAR_Utilities).

`sub_nondivergentWind.f90`: compute the nondivergent component of the wind (Dynamics).

`sub_openObsFiles.f90`: open a set of observational files (TopLevel).

`sub_pvInvert.f90`: apply Chris Davis' piecewise PV inversion (Vorticity).

`sub_qvec.f90`: compute Q-vectors and Q-vector divergence of the flow (Dynamics).

`sub_stats.f90`: compute the statistics (mean, variance, etc) of the field (Math).

`sub_steeringFlow.f90`: compute the steering flow of a vortex (Dynamics).

`sub_tcBogus.f90`: implant a synthetic tropical cyclone vortex in the field (Dynamics).
`sub_tcRemove.f90`: remove an existing vortex from the flow (Dynamics).

`sub_trajectory.f90`: compute the trajectory of a parcel

`sub_uvRotate.f90`: rotate winds to/from grid/north relative (Dynamics).
`sub_validTimes.f90`: return a timer-sized array (dimensionned [ntimes,3] with the valid times of the date stamps held in the provided timer (TopLevel).
`sub_vorticityComponents.f90`: compute shear and curvature vorticity components (Vorticity).

In addition, modification of the I/O subprograms may be necessary in special circumstances. The high-level I/O interface is contained in the following subprograms.

`sub_readGrid.f90`: read a data field (FileIO).

`sub_readObs.f90`: read a given element from an observational file (FileIO).

`sub_writeGrid.f90`: write to a data field (FileIO).

5.7 Special Topics

This section outlines some of the special features available throught the SPA. Many of these functionalities rely on installed packages whose availability may vary from platform to platform.

5.7.1 Plotting with the SPA

The SPA provides a basic interface to the powerful NCAR Graphics plotting package. A set of high-level interfaces to NCAR Graphics are provided with the SPA distribution, including scalar and vector plotting routines and a tropical cyclone track mapping algorithm that uses best track data from a variety of sources. Each of the calls to the NCAR Graphics interface routines has numerous optional arguments, only a few of which will be described

here. Use the `spainfo` utility to get version-specific interfacing information for each of the subprograms described here. The default operation of each of these subprograms is to produce an NCAR metafile viewable with the `idt` program (provided with the NCAR Graphics package). The `ctrans` program (also provided with NCAR Graphics) can be used to convert the metafile to a printable PostScript file.

Each of the primary plotting interfaces described in the following paragraphs has a set of optional arguments for controlling the superposition of fields and the advancement of the plotter. In each case, setting `initialize=.true.` (the default) results in the generation of a background map (generally only done once for each plot); setting `advance=.true.` (the default) causes the plotter to advance after laying down the requested field (set `advance=.false.` for superposition); and setting `shutdown=.true.` (the default) closes the active NCAR Graphics process. An example is provided below in which these arguments are used to produce a plot containing multiple fields.

The `drawPlot` (function) interface is used for producing plots of scalar fields. After use-associating the `NCAR_Utilities` module, a call to `drawPlot` can be made directly to produce a metafile containing a plot of the desired field. Numerous optional arguments are available to help the user tune the plot to exact requirements (for example, generating a colour-filled plot rather than contour lines). Note that several optional configuration file flavours can be provided to the `drawPlot` subprogram to ease the generation of multiple plots with different configurations. For a complete description of the `drawPlot` arguments, use the `spainfo` utility.

The `drawVectors` (function) interface is used for producing plots of a vector field. As described above, this subprogram can be invoked any time after the NCAR Graphics module is associated and produces a standard metafile. Both wind barbs and arrows can be generated and configured using this interface. As for the scalar plotting interface, either a configuration file or a series of optional arguments can be used to describe the nature of the vectors to be plotted.

The `drawText` (function) interface allows the user to add text to the map or page. Note that the `coord` optional argument allows the user to distinguish between map and viewport interpretation of the given positional coordinates.

The `drawLine` (function) interface allows the user to add a line (or set of connected lines) to the map or page. Note that the `coord` optional argument allows the user to distinguish between map and viewport interpretation of the given positional coordinates. Also, lines can be forced to lie along latitude/longitude circles rather than appearing as straight on the plotted projection.

The `drawCentered` (function) interface allows the user to generate storm-centered plots in cylindrical coordinates. In this case, the `centre` argument defines the coordinates of the point to which the pole of the globe is rotated. The result is a polar steregraphic (cylindrical coordinate) grid centred on the location of interest, usually the centre of a system. Because the calculations used to compute the range rings on the rotated polar grid are contained in NCAR Graphics, the computational function `ncarCentered`, which produces the gridded cylindrical coordinate data, is also a part of the NCAR Graphics API.

The `drawTracks` (function) interface allows the user to overlay tracking information on scalar and vector plots. The datafile named in the input argument list (use `spainfo` for

details) should be of the form generated by the `trackinfo` utility provided with this package (see Section 4.5 [trackinfo], page 14).

The `drawTrajectory` (function) interface allows the user to generate backwards and forwards parcel trajectories. As described above, this subprogram can be invoked any time afer the NCAR Graphics module (`NCAR_Utilities`) has been use associated and produces a standard metafile. The trajectory can be colour coded for pressure levels (the default) or for a user-defined field as provided by the optional argument. Numerous options for highly-accurate sub-trajectory computations and start/end plotting symbols are avialable through optional arguments. Note that all gridded fields are provided as four dimensional arrays. The fourth dimensions is time, so all data should be read into the data array before `drawTrajectory` is called. This means that `drawTrajectory` is one of the very few SPA utilities that must be called from outside the main time loop of the driving program.

This example code segment shows the set of calls required to produce a colour-filled contour plot with vector (barb) overlays. A limited optional argument list is provided to each plotting subprogram and is by no means exhaustive for each.

```
GKSerror = drawPlot(theta,dim,                &
      mapBounds=(/-90.,-180.,90.,180./),      &
      projection='CE',                        &
      centre=(/0.,0./),                       &
      fillColours=.true.,                     &
      colourMap=.false.,                      &
      initialize=.true.,                      &
      shutdown=.false.,                       &
      advance=.false.)
 GKSerror = drawVectors(u,v,dim,              &
      vectorRGB=(/1.,1.,0./),                 &
      mapBounds=(/-90.,-180.,90.,180./),      &
      projection='CE',                        &
      centre=(/0.,0./),                       &
      colourMap=.false.,                      &
      initialize=.false.,                     &
      advance=.true., shutdown=.true.)
```

The `ncarInit` and `ncarEnd` (functions) are low-level interfaces not usually called directly by the user. They are, however, used heavily by the drawing interfaces described above. Direct calls to these subprograms are, however, particularly useful if PostScript files are to be generated instead of metafiles (for example, if a large number of plots are to be produced by a single program). For a complete description of these interfaces, use the `spainfo` utility.

To produce direct PostScript output, the following would have to be added above the plotting calls in the example above.

```
GKSerror = ncarInit(device='ps',              &
      orientation='portrait',                 &
      outFileName='testPlot.ps')
```

And the following would have be added after the end of the segment in the plotting example.

```
GKSerror = ncarEnd(GKSerror)
```

This would ensure the correct startup and shutdown of the NCAR Graphics processes required to generate the requested PostScript file (in this case rather unimaginatively called `testPlot.ps`).

Appendix A Copying Conditions

A.1 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related

matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at

your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warrany Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.1.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled
``GNU Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

A

absolute vorticity 22
advection 22
advection calculation mode, metcal 8
anomaly correlation 22
APE 25
area average 22
area, fill, NCAR Graphics 23
available potential energy 25
average, horiztonal 22
average, vertical 22
averaging mode, metcal 8
axis of contraction 22
axis of dilitation 22

B

balance, geostrophic 22
balance, gradient 22
balance, hydrostatic 24
basic state, Eady 25
below-ground values, fill 25
bogus, vortex 26
Brundt Vaisala frequency 22
budget, available potential energy 25
budget, eddy kinetic energy 25
BURP files 2

C

CAPE 25
case, lower 24
case, upper 25
centereing, storm, NCAR Graphics 23
centering, storm 26
centre, TC 25
CIN 25
clock, system 24
close observational files 25
coding, customized 16
collection, garbage 25
colour map reset, NCAR Graphics 25
colour map, NCAR Graphics 22
column integral 22
comp90.d 19, 20, 21
components, vorticity 26
components, wind, x/y 25
contour plot, NCAR Graphics 23, 27
contraction, axis of 22
convergence, moisture 24
conversion, date 22
Coriolis force 22
coupling index 22
creating a interface 19
creating a subprogram 21
creating a template 16
creating an interface 21
creation, timer 25
curvature vorticity 26
customized coding 16
cyclone tracking mode, metcal 9

D

data files, GEMPAK 2
data files, RPN 2
data formats 1
date conversion 22
date stringification 22
date, Julian 24
dates processing 17
deformation 22
density 22
derivative, horizontal 22
derivative, second 22
derivative, vertical 22
developer's interface 6
developer's interface 16
developer's interface, example 19, 20, 21
developer's interface, metcal 11
dewpoint 23
dewpoint depression 23, 24
difference calculation mode, metcal 8
differencing 23
digital filter 23
dilitation, axis of 22
direction, wind 25
dissipation, power 24
distance 24
divergence 23
divergence of Q-vectors 26
dynamic tropopause 23

E

Eady atmosphere 25
Eady basic state 25
eddy APE 25
eddy EKE 25
EKE 25
equivalent potential temperature 25
equivalent potential temperature, inversion 23
Ertel potential vorticity 23
exponential 23
extending, metcal interface 10
extending, spa 6

F

FDL, GNU Free Documentation License 30
file IO page, metcal 7
file management, fst2sig 14
file management, splitFST 14
file size 24
file size, observational 24
file, grid, read 26
file, grid, write 26
file, observational, read 26
files, observational, close 25
files, observational, open 26
fill area, NCAR Graphics 23
fill below-ground values 25
fill missing values 25
filter, smoothing 25
filter, time 23
filtering mode, metcal 8
frontogenesis, Miller 25
fst2sig 14
full file processing 17
func_timeLoop.f90 17
func_timeLoop.f90, dates processing 17
func_timeLoop.f90, file processing 17
func_timeLoop.f90, list processing 18
func_timeLoop.f90, range processing 18

G

garbage collection 25
GEMPAK data files 2
geopotential 22
global grid 24
graphics 26
grid file, read 26
grid file, write 26
grid relative winds 26
grid, global/regional 24
gridpoint locator 25

H

horizontal average 22
horizontal derivative 22
horizontal interpolation 24
humidity, realtive 24
humidity, relative 23
humidity, specific 23, 24
hurricane summary, NCAR Graphics 23
hurricane tracking 14
hydrostatic balancing 24

I

in-depth, metcal 7
index, level 25
initialization, vortex 26
initialize, NCAR Graphics 24
input data 1
installation, spa 3
integral, vertical 22
intensity, potential 24
interface, creating 19, 21
interface, developer's 6
interface, developers, metcal 11
interpolation mode, metcal 9
interpolation, horizontal 24
interpolation, isentropic 24
interpolation, vertical 25
introduction, metcal 1, 7
introduction, spa 1
inverse Laplacian 24
inversion, equivalent potential temperature 23
inversion, potential vorticity 26
irrotational wind 26
isentropic potential vorticity 24
isentropic variable 24

J

Julian day 24

K

kinetic energy 25

L

Laplacian 24
Laplacian, inversion 24
LCL 25
length, path 24
level index 25
LFC 25
line drawing, NCAR Graphics 23
line, NCAR Graphics 27
list processing 18
list, SPA subprograms 22
location, gridpoint 25
lower case 24

M

major modes, metcal 7
map scale 24
map, NCAR Graphics 24
maximum potential intensity 24
mean 26
mean, running 25

memory, reducing usage 25
metcal, advection calculation 8
metcal, averaging 8
metcal, cyclone tracking 9
metcal, developer's interface 11
metcal, difference calculation 8
metcal, file IO page 7
metcal, filtering 8
metcal, in-depth 7
metcal, interface, modifying 10
metcal, interpolation 9
metcal, introduction 1, 7
metcal, major modes 7
metcal, modifying 10
metcal, overview 1
metcal, simple math calculation 8
metcal, sounding 9
metcal, standard outputs 8, 10
Miller frontogenesis 25
missing values, fill 25
modes, metcal, advection calculation 8
modes, metcal, averaging 8
modes, metcal, cyclone tracking 9
modes, metcal, difference calculation 8
modes, metcal, filtering 8
modes, metcal, interpolation 9
modes, metcal, major 7
modes, metcal, simple math calculation 8
modes, metcal, sounding 9
modes, metcal, standard outputs 8, 10
modifying a subprogram 20
modifying an interface 20
modifying, metcal interface 10
modifying, spa 6
modules, trackinfo 14
moist potential vorticity 25
moisture convergence 24
mositure transport 26

N

NCAR Graphics 26
NCAR Graphics, colour map 22
NCAR Graphics, colour map reset 25
NCAR Graphics, contour plot 23, 27
NCAR Graphics, draw line 23
NCAR Graphics, fill area 23
NCAR Graphics, hurricane summary 23
NCAR Graphics, initialize 24
NCAR Graphics, line 27
NCAR Graphics, map 24
NCAR Graphics, page layout 26
NCAR Graphics, panels 26
NCAR Graphics, pinwheel plot 23
NCAR Graphics, shutdown 24
NCAR Graphics, storm centering 23, 27
NCAR Graphics, text 23, 27
NCAR Graphics, track map 23, 27
NCAR Graphics, trajectory analysis 23, 28
NCAR Graphics, vector plot 23, 27
nondivergent wind 26
north relative winds 26

O

observational data, BURP format 2
observational file size 24
observational file, read 26
observational files, close 25
observational files, open 26
omega 25
open observational files 26
order 23
overview, metcal 1
overview, SPA 1

P

pad whitespace, left 24
page layout, NCAR Graphics 26
panels, NCAR Graphics 26
path length 24
penetration depth, Rossby 24
performance 24
pinwheel plot, NCAR Graphics 23
plotfst 13
plotting 26
plotting, tracks 14
potential intensity 24
potential temperature 25
potential temperature, equivalent, inversion 23
potential vorticity inversion 26
potential vorticity, Ertel 23
potential vorticity, isentropic 24
potential vorticity, moist 25
power 23
power dissipation 24
predictability, anomaly correlation 22
pressure, sea level, reduction 25
program, fst2sig 14
program, metcal 7
program, plotFST 13
program, splitFST 14
purpose, spa 4

Q

Q-vectors 26
Q-vectors, divergence 26

R

range processing ... 18
read grid file ... 26
read observational file ... 26
reducing memory usage ... 25
reduction, sea level pressure ... 25
regional grid ... 24
relative humidity ... 23, 24
removal, vortex ... 26
Rossby penetration depth ... 24
rotate winds ... 26
RPN standard files ... 2
running mean ... 25

S

scale, map ... 24
sea level pressure reduction ... 25
second derivative ... 22
separation ... 24
shear vorticity ... 26
shutdown, NCAR Graphics ... 24
simple math calculation mode, metcal ... 8
skill score, anomaly correlation ... 22
smoothing filter ... 25
sounding mode, metcal ... 9
SPA subprogram list ... 22
spa, extending ... 6
spa, in-depth ... 3
spa, installation ... 3
spa, introduction ... 1
SPA, overview ... 1
SPA, plotting ... 26
spa, purpose ... 4
spa, structure ... 5
spaextract ... 20, 21
spainfo ... 19
spalist ... 19
spaopen ... 16, 19
specific humidity ... 23, 24
speed, wind ... 25
splitFST ... 14
stability, Brundt Vaisala frequency ... 22
stability, Rossby penetration depth ... 24
stability, static ... 25
standard deviation ... 26
standard outputs mode, metcal ... 8, 10
static stability ... 25
statistics, mean ... 26
statistics, standard deviation ... 26
statistics, T-test ... 25
steering, vortex ... 26
stepping, time ... 17
storm centering ... 26
storm centering, NCAR Graphics ... 23, 27
streamfunction ... 22, 25
stringification, date ... 22
structure, spa ... 5
Student's T-test ... 25
subprogram, creating ... 21
subprogram, modifying ... 20
system clock ... 24

T

T-test ... 25
TC, bogus ... 26
TC, centre ... 25
TC, removal ... 26
TC, steering ... 26
temperature, dry bulb ... 25
temperature, equivalent potential ... 25
temperature, potential ... 25
template, creation ... 16
template, example ... 19, 20, 21
text, NCAR Graphics ... 23, 27
time filter ... 23
time loop ... 25
time looping ... 17
timeLoop, dates processing ... 17
timeLoop, file processing ... 17
timeLoop, list processing ... 18
timeLoop, range processing ... 18
timer, creation ... 25
times, valid ... 26
tools, other ... 11
track map, NCAR Graphics ... 23, 27
trackinfo.py ... 14
tracking ... 25
tracking, hurricane ... 14
trajectory ... 26
trajectory analysis, NCAR Graphics ... 23, 28
transport, moisture ... 26
trim whitespace, left ... 24
tropopause, dynamic ... 23

U

upper case ... 25

V

valid times ... 26
vector plot, NCAR Graphics ... 23, 27
velocity potential ... 25
vertical average ... 22
vertical derivative ... 22
vertical integral ... 22
vertical interpolation ... 25
vertical motion ... 25
vortex initialization ... 26
vortex removal ... 26
vortex steering ... 26
vorticity components ... 26
vorticity, absolute ... 22

W

whitesapce, pad, left 24
whitespace, trim, left 24
wind, direction 25
wind, irrotational 26
wind, nondivergent 26
wind, rotate 26
wind, speed 25
wind, streamfunction 25
wind, velocity potential 25
wind, x/y components 25
write grid file 26