Forecast Production Assistant Version 8

Administrator's Manual

Contents

GNU Free Documentation License

Version 1.3, 3 November 2008

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 noncommercially. 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.

The "publisher" means any person or entity that distributes copies of the Document to the public.

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 in 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 Warranty 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 under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

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 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. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Introduction

The Forecast Production Assistant (FPA) is a unique computer program developed by the Meteorological Service of Canada for use in operational weather forecasting.

The FPA allows forecasters to represent and interact with conceptual models of the atmosphere. Meteorologists use the FPA to create a series of weather charts, known as depictions, that show weather fields over a sequence of times. The weather fields can contain grid point data or sets of areas, lines or scattered points. The weather depictions are not just graphical displays, but windows into an object database. By connecting processes to the database, forecast offices can automatically generate forecast products or run local models, while spending more of their time on weather analysis, diagnosis and prognosis.

About This Guide

The Forecast Production Assistant Administrator's Guide is intended for the person responsible for installing, configuring, updating and maintaining the system.

This guide assumes that you have a working knowledge of Linux.

Chapter 1

System Requirements

This section details the hardware and software requirements for running FPA.

1.1 Hardware

- A PC workstation capable of running the software below.
- Any graphics card which provides 8 bits of colour and 1280 x 1024 resolution.
- The disk space requirements are: 15Mb per version of the required executable software, up to 10Mb per version for optional software and 5Mb per database. Thus the disk space required for a minimal system is 20Mb and for a system with two databases and all optional software installed, 35Mb.
- Recommended minimum of 96 Megabytes RAM.
- A two or three button mouse and/or a graphics tablet with at least a two button stylus and/or puck (tablet with puck is highly recommended for smooth drawing).

1.2 Software

- Linux (Ubuntu, Debian, RedHat, Centos)

Chapter 2

Understanding How FPA Works

This chapter provides an introduction to the functionality of the FPA software. It is not intended to be detailed but rather an overview to give you an idea of what is happening.

2.1 Download and ingest

FPA uses its own internal data representations, but provides procedures to ingest GRIB format files.

For new Canadian clients, you may manually download up-to-date GRIB files. However, arrangements should be made for an automatic download with CMC as soon as possible. Clients outside the Meteorological Service of Canada will have their own source of data. As long as the data is in GRIB format it should be compatible with the ingest software.

The ingest daemon **fpaingest** detects the presence of new data files, and starts an ingest program.

There are two programs available for ingesting GRIB data: **gribin** and **gribin2**. These programs process the individual fields in each GRIB file, re-interpolate to the local map projection and deposit the result to the appropriate guidance directories (see Directories Setup, (Section 5.5)).

gribin
: Can process GRIB editions 0 and 1, and requires a `Gribs` configuration file. (see Gribs, (Section 6.4))

gribin2
: Can process GRIB editions 0, 1 and 2, and requires a `Ingest` configuration file. (see Ingest and Ingest.***name***, (Section 6.5))

The **fpaingest** daemon, can start user defined ingest programs as well.

2.2 The Depiction Editor

The Depiction Editor (**xfpa**) is used to create and manage a set of graphical weather depiction charts. The depictions are intended to represent a time sequence that describes the predicted behaviour of certain weather elements during the given period.

The Depiction Editor provides several tools for modifying the weather elements contained in each depiction. It also provides a facility for examination and rudimentary analysis of operational model guidance forecasts in chart form. This assists the forecaster in predicting the behaviour of the various weather elements.

The Depiction Editor also incorporates a mechanism for connecting specific weather features through time, so that it can interpolate the depiction sequence onto a shorter interval. This results in a related sequence of interpolated depictions, which can be used to create forecast products at intermediate times.

Please refer to the **`User's Manual`** for details on the functionality of the FPA Depiction Editor.

The **xfpa** command can be run from the command line with a number of arguments:

`xfpa` [-askForTimes] [-profile ***`name`***] [-s[etup] ***`filename`***] [-stateDir ***`directory`***] [-t0 ***`YYYY:JJJ:HH`***] [-v[isible] ***`key`***] [+viewerMode]

Table 2.1: Explanation of **xfpa** optional arguments

arg...	**meaning...**
-askForTimes	Puts up a dialog box asking which range of depiction times are to be read into the FPA
-profile ***`key`***	Start using the specified profile instead of putting up a dialog asking the user for a profile. "none" is a valid input.
-s[etup] ***`filename`***	Specifies the name of the file to use as the setup file. This may be an absolute path name or relative to the default setup directories.
-stateDir ***`directory`***	The directory to use to store the state store file. This is only used if in viewer mode.
-t0 ***`YYYY:JJJ:HH`***	Specifies the time to use as T0 for the depiction sequence. ***`YYYY`*** = 4 digit year, ***`JJJ`*** = julian day of the year, ***`HH`*** = hour of the day
-v[isible]***`key`***	Bring up FPA with fields visible as specified by the block of data indicated by the ***`key`*** in the preset field visibility config file.
+viewerMode	Runs the FPA in viewer mode. All loading, deletion, saving and editing commands are not available so that only sequence viewing is possible.

Note

The FPA restricts access to the graphical weather depictions to one user at one time. The FPA enforces this restriction by creating a `.LOCK*` file in the `Depict` directory while **xfpa** is running. Attempts to access the same graphical weather depictions with another **xfpa** command will result in an error.

Note that Guidance and Allied Model directories can be shared, and the graphical weather depictions can be accessed by other users, but only one user can add, delete or modify graphical weather depictions at one time.

One result of this restriction is that the FPA code assumes that no other external process will add, delete or modify graphical weather depictions (that is, files in the `Depict`, `Backup` and `Interp` directories) while **xfpa** is running, and any external process trying to do so may result in errors in the FPA, up to and including crashes!

2.3 Product Generators

There are several different product generators. Each is designed for producing a different type of product out of the depiction sequence. One of these generators is invoked when the **Generate** button is selected.

The Graphics Product Generator application allows users to create tailored graphical products or simpler ASCII products from the FPA database. The two graphical product applications are PSMet (producing output in PostScript format) and SVGMet (producing output in SVG format, an open XML format). The ASCII application is TexMet. All three applications use a command language in product generation files to sample the FPA database and display the results in a number of different formats. Please refer to the **`Graphics Product Generator Reference Manual`** for details on the functionality of the applications and their product definition files.

There is also an internal graphics application for copying FPA metafiles to transfer them to another location where they can be used as initial fields or as guidance. Note that the format of product definition files for copying FPA metafiles is discussed in Product Definition File Formats, (Appendix D).

2.4 Product Editors

Certain text and graphical products have a corresponding product editor to allow the user to review and modify the product if necessary. The appropriate editor is invoked when the Edit button is selected from within the product generation menu.

2.5 Allied Models

A number of useful models have been integrated with FPA. We use the term "allied" to indicate that they are not actually part of FPA itself. However, a mechanism exists that allows users to integrate an Allied Model in to the FPA, so that the Allied Model can be run directly from the FPA interface.

Integrating an Allied Model into the FPA is accomplished by creating one or more specialized modules to: extract information from the FPA database required as input by an Allied Model; to run the Allied Model; or to translate the output of an Allied Model into FPA database files so as to import the Allied Model information back into the FPA depiction sequence. Allied Models are described in Adding New Allied Models, (Appendix E).

The last sections of Adding New Allied Models, (Appendix E) describe special internal Allied Models called "FPA Warp", "FPA Create Area" and "FPA Create Contour". These Allied Models can be used to create metafiles without the need to develop special code, by simply adding lines in the Local Setup file and the Local Configuration file.

"FPA Warp" uses the fpawarp program to merge point data (either over a grid or at random locations) with an FPA field, modifying the field to match the point data values.

"FPA Create Area" uses the fpacreate_area program to generate Discrete areas from the attributes of features in Line, Link Chain or Scattered type fields.

"FPA Create Contour" uses the fpacreate_cont program to generate Discrete areas from the contours of Continuous fields.

Chapter 3

Preparing to Install FPA Source Code

3.1 Preparing to Compile FPA Source Code

3.1.1 Operating System

This version of FPA has been successfully compiled and run on Ubuntu 16.04 and Centos 7.

3.1.2 Source Code

Copy the FPA master directory (`fpav8`) to your preferred working directory.

Note
We suggest you create a dedicated user account for FPA development (i.e. fpadev); create your master directory in the home directory of this user account (/home/fpadev/fpav8); and point your FPA user accounts there. (See Example 3.7.)

The source code is available on http://github.com. To install the source code use the following commands:

```
apt-get install git-core  (for Ubuntu or Debian)
  or
yum install git-all  (for Centos or RedHat or Fedora)

git clone https://github.com/emmalhung/xfpa.git fpav8
```

Note
Users who would like to contribute to the FPA project are encouraged to get a github account and create a fork of the FPA source code.

All FPA software and scripts depend on the existence of the environment variable, `FPA` which contains the full path of the FPA master directory. Edit the environment profile file that corresponds with the normal

login shell for the current account, and add one exported environment variable called `FPA`, which has a value equal to the full path of the FPA master directory. Then add `$FPA/bin` and `$FPA/sbin` to the `$PATH` variable.

The profile file name will depend on the flavour of Linux shell you are using. Some examples are `.profile`, `.csh.login`, `.vueprofile`, `.bash_profile`.

Example 3.1 Sample (bourne shell) development profile file

```
FPA="/home/fpadev/fpav8"
PATH="$FPA/bin:$FPA/sbin:$PATH"
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
  PATH="$HOME/bin:$PATH"
fi
export FPA PATH
```

Note
You should log out and back in again in order for these changes to take effect.

There are several scripts (ones that are expected to run outside of a user's shell) that set `FPA` to `/home/fpadev/fpav8` you should edit these files and set `FPA` to your preferred master directory.

- $FPA/etc/d.fpaingestrc
- $FPA/etc/d.fpalaunch
- $FPA/etc/d.fpaschedrc

Other scripts hardcode the fonts directory:

- $FPA/etc/d.fparc
- $FPA/bin/putfonts
- $FPA/bin/truetypefonts_[on|off]

3.1.3 Dependencies

3.1.3.1 Required packages for Ubuntu or Debian Linux

Use **apt-get install** to install each package and its dependencies

- libxml2-dev
- libmotif-dev
- gfortran
- libpng-dev
- libjpeg-dev
- libtiff-dev
- libX11-dev
- libxt-dev
- libxft-dev
- libxpm-dev
- libxmu-dev
- libgrib2c-dev (optional to run GRIB ingest)
- libjasper-dev (optional to run GRIB ingest)
- dblatex (optional to compile documentation)

Note
You must be logged in as root in order to run the apt-get package manager

Example 3.2 Using 'apt-get install'

```
~# apt-get install gcc
```

Example 3.3 How to track down missing libraries using 'apt-file'

```
~# apt-get install apt-file # Install apt-file
 ...
~# apt-file update # Update the database
 ...
~# apt-file search /usr/lib32/libpng.so
ia32-libs-dev: /usr/lib32/libpng.so
```

3.1.3.2 Required packages for Centos or RedHat or Fedora Linux

Use **yum install** to install each package and its dependencies

- libxml2-devel
- openmotif-devel
- gcc-gfortran
- libpng-devel
- libjpeg-devel
- libtiff-devel
- libX11-devel
- libXt-devel
- libXft-devel
- libXpm-devel
- libXmu-devel
- epel-release (optional to run GRIB ingest)
- g2clib-devel (optional to run GRIB ingest)
- dblatex (optional to compile documentation)

Note
You must be logged in as root in order to run the yum package manager

Example 3.4 Using 'yum install'

```
~# yum install gcc
```

Example 3.5 How to track down missing libraries using 'yum whatprovides'

```
~# yum whatprovides /usr/lib32/libpng.so
ia32-libs-dev: /usr/lib32/libpng.so
```

3.1.4 Compiling Source Code

Compiling FPA source code involves an intricate web of Makefiles. The customary **configure; make** sequence is replaced by a script called **fpamake**. This script will start from the current working directory and recurse to any subdirectories. **fpamake - all** will start from the master directory and recurse through all subdirectories. The file **fpamake** is located in `$FPA/sbin`. Watch the output from the compile closely to ensure that all components are compiled correctly. You will want to check the log file for any errors. It will be located in the master directory `$FPA`, and the name will start with the word **`log`** and end with the value returned by **platform**.

Ensure that all the Dependencies, (Section 3.1.3) are installed.

From the `$FPA` directory run **fpamake all**.

After the initial build you can use the command **fpamake - all** to compile after making changes to the source code.

3.1.5 Trouble Shooting Compilation

There is a script called **platform** that tries to determine what platform you are running. If you run it and do not get something appropriate, then you may need to edit the script to correct its behaviour. The **platform** script is located in `$FPA/bin`.

3.1.6 The codeword

The open source version of FPA contains an unlimited license codeword located in `$FPA/license/Codes`. This codeword is not compatible with older versions of FPA

3.1.7 Test the FPA software

To test the FPA software, Run the FPA depiction editor, as in: **xfpa** or **xfpa -s *`setup_file`***. Where ***`setup_file`*** is the name of the setup file to be used. If omitted, the default setup file is used (see .fparc below).

If you do not have a database, you can use a demo database located in `$FPA/demo/`.

Example 3.6 Using a demo database to test FPA

```
>cd $HOME
>tar -xzf $FPA/demo.tgz
>xfpa -s av_atlantic
```

You should see the FPA window appear. To exit from the FPA depiction editor: select **Field** → **Quit** on the menu bar, near the upper left; a confirmation requester will then appear; select **Yes** and the FPA depiction editor will terminate.

3.2 Installation/update of FPA databases

You may wish to update your existing FPA databases and/or install one or more new databases. It is intended that an FPA database should be administered from a single user account. However, one user can administer several databases.

The FPA master directory must be accessible (i.e. has been installed or remote mounted). Repeat the following steps for each FPA database you wish to install or update.

1. If necessary, create the administrator account for this FPA database.
2. Log in to the administrator account for this FPA database.
3. If necessary, update the local environment profile file. All FPA software depends on the existence of the environment variable, `FPA`, which contains the full path of the FPA master directory. Edit the environment profile file (`.profile`, `.csh.login`, `.vueprofile`, `.bash_profile`, etc) that corresponds with the normal login shell for the administrator account, and add an exported environment variable called FPA, which has a value equal to the full path of the FPA master directory. If you created a user account for FPA development and the source code is located in a directory called `fpav8` then you might edit your profile file as follows:

Example 3.7 Sample (bourne shell) user profile file

```
FPA="/home/fpadev/fpav8"
PATH="$FPA/bin:$PATH"
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
  PATH="$HOME/bin:$PATH"
fi
export FPA PATH
```

4. When done, re-invoke the new profile by "sourcing" (dotting) it or by logging out and back in.
5. For each database controlled by this administrator account, build or update the database and setup file.
   i. Execute the **mkfpadb** script. This script first builds or updates the FPA environment "dot" file, `$HOME/.fparc`. If this file already exists, and has not yet been updated, **mkfpadb** attempts to preserve all your customizations. The old version is saved for comparison.
   ii. Next, **mkfpadb** asks for the setup directory and the name of a setup file. A new setup file is created, using the template setup file as a model. If the given setup file exists, the map projection information and database directory are preserved, and a copy of the original is made for comparison.
   iii. This script asks you for the data directory name, or allows you to redefine the data directory if the setup file already exists. It then builds or updates the database, creating various data subdirectories if needed.
   iv. You must obtain a `base_map.fpa` file and place it in the `$HOME/your_database/Maps` directory to display a background map in the field editor.

Note
You can find a number of example FPA map files in the directory `$FPA/data/common/Examples`.
TODO: There was a tool that would translate ESRI Shapefiles into FPA map metafiles, it is now out of date. Somebody with ArcGIS programming experience might want to try replace it.

If you chose to name your background map something other than `base_map.fpa` you will need to edit your setup file (the "[map.base]" section in the 'interface' block) to reflect this.

v. Customize the setup file that was built for you. The new setup file contains instructions for customizing. Lines beginning with "#!" describe sections that will generally have to be modified. If you updated an existing setup file, there is no guarantee that all your settings will be preserved reliably. Use the retained copy of the original for comparison.

vi. Follow the customization instructions in the file `$HOME/.fparc`. This file is sourced ("dotted") by all FPA applications, to set a number of environment variables to control such things as where to search for the setup file, where to send log output and where to send printer output. If updating several databases, this file is modified only the first time, and all existing variable settings will have been preserved. In particular, you may wish to supply the name of the default setup file (`FPA_SETUP_FILE`). This should probably be the name of one of the setup files that was built in the previous steps. The local destination for GRIB files can also be specified (`FPA_LOCAL_GRIB`). This specifies the default directory where the GRIB files are to be found by the ingest process.

3.2.1 Test the FPA database

To test your new FPA database: run the FPA depiction editor, **xfpa [-s *setup_file*]**. Where ***setup_file*** is the name of the setup file for the database you just created. If omitted, the default setup file is used (see .fparc above).

You should see the FPA window appear. It will hopefully be displaying the correct map background. If you just installed this database, there will be no depictions in the sequence. To exit from the FPA depiction editor: select **Field** → **Quit** on the menu bar, near the upper left; a confirmation requester will then appear; select **Yes** and the FPA depiction editor will terminate.

3.3 Setting up the automatic ingest

Follow these steps to set up global control of the automatic FPA ingest daemon. This daemon can be used if you are receiving an automatic download of GRIB fields from a supplier of your choice. Even if you are not receiving an automatic download, you may still set up the automatic ingest daemon to trigger whenever you manually download the GRIB files. The ingest daemon monitors the modification times of a predefined set of GRIB files in a given directory. If any of these files is "updated", then the appropriate ingest is launched. If you do not setup the automatic ingest, you will have to manually launch the ingest program itself (see below), each time new data arrives.

Note
An automatic ingest is not needed if you plan to run FPA applications only in a demonstration mode, on "pre-canned" data.

Note
The following approach was developed many years ago, and modern Linux distributions have helper functions that could be used to automate much of the following process.
For Ubuntu or Debian ... man update-rc.d
For Centos or RedHat or Fedora ... man chkconfig

1. Login as the user who owns the `$FPA` directory.
2. Next set up the ingest control for each FPA database. Repeat the following steps for each FPA database administrator account that will require an automatic ingest.
   i. Log in (using su(1)) to the administrator account for this FPA database, as in: **su - *user***
   ii. As delivered, the environment "dot" file `$HOME/.fparc` defines a variable, `$FPA_LOCAL_GRIB`, which defines the directory where GRIB data is found. As delivered, your setup files define the "ingest.src" directory to refer to `$FPA_LOCAL_GRIB`. This arrangement will work fine if all your GRIB data is in the same directory. If this is not the case, then you can define a new directory tag in the "directories" block and set it as you need. (Directories Setup, (Section 5.5).)
   iii. Copy and rename the ingest control "dot" file into the home directory as in:
   ```
   >cp $FPA/etc/d.fpaingestrc $HOME/.fpaingestrc
   ```
   This file is normally sourced ("dotted") on boot-up, by the file `$FPA/etc/fpaschedrc`, to restart the ingest daemon for this database. Edit the file `$HOME/.fpaingestrc` and add a line to start up the ingest daemon for each database setup file name. Customization instructions are found inside the file itself. Note that each ingest requires a different setup file name. However, it is possible that more than one setup file could access the same FPA database. In this case, care must be taken that the file names for the "log" and "states" files in the "ingest" block of each setup file are not the same!
   iv. Log out (you should then still be logged in as the fpa administrator).
3. Edit the ingest schedule file `$FPA/etc/fpaschedrc` to provide a startup line for each FPA administrator account that requires an automatic ingest. Instructions are found inside the file itself.
4. The ingest schedule file `$FPA/etc/fpaschedrc` should normally be invoked on boot-up, so that the automatic ingest(s) will be restarted. To do this, first login as root then copy the file `d.fpalaunch` to fpalaunch in the appropriate directory. The copy should be owned by root:root and should have read and execute permission for all. The file should be copied to the directory `/etc/init.d`, for example:
   ```
   >cp $FPA/etc/d.fpalaunch /etc/init.d/fpalaunch
   >cd /etc/init.d
   >chown root:root fpalaunch
   >chmod 755 fpalaunch
   ```

5. Edit the copied file, `fpalaunch`, as appropriate for your site. Find further instructions in the file itself.

6. Next you must add symbolic links to the `/etc/rc[0123456].d` directories.

```
>ln -s /etc/init.d/fpalaunch /etc/rc0.d/K050fpalaunch
>ln -s /etc/init.d/fpalaunch /etc/rc1.d/K050fpalaunch
>ln -s /etc/init.d/fpalaunch /etc/rc2.d/S950fpalaunch
>ln -s /etc/init.d/fpalaunch /etc/rc3.d/S950fpalaunch
>ln -s /etc/init.d/fpalaunch /etc/rc4.d/S950fpalaunch
>ln -s /etc/init.d/fpalaunch /etc/rc5.d/S950fpalaunch
>ln -s /etc/init.d/fpalaunch /etc/rc6.d/K050fpalaunch
```

   To start the ingest for the first time, execute fpalaunch as root.

```
>/etc/init.d/fpalaunch start
```

   or reboot the computer to make sure fpalaunch starts automatically.

3.4 Adding another automatic ingest

You can add another ingest without stopping the ingest scheduler. However, you should add the new ingest to the ingest control "dot" file so that the new ingest will start on re-boot.

1. Edit the file `$HOME/.fpaingestrc` and add a line for the new setup file name.

2. Start up the new ingest

```
>fpaingest start setup_file
```

   where ***setup_file*** is the name of your setup file.

3. Check the ingest status

```
>fpaingest status
```

3.5 Modifying setup files for automatic ingests

It is important to realize that the setup file for an ingest is read only once, the first time an ingest for an FPA database is started. Any subsequent changes to the setup file that could effect an automatic ingest will not take effect, until the ingest is stopped and restarted, as in:

```
>fpaingest stop setup_file
>fpaingest start setup_file
```

where ***setup_file*** is the name of your setup file

3.6 Manual Ingest

If, for some reason, you do not wish to use the automatic ingest, you will need to perform ingests manually. Although some fields may be drawn from scratch, the true power of FPA comes from its ability to start with existing model-based fields. At present, fields from an external model can only be accessed in GRIB format.

Use **gribin** or **gribin2** to ingest the new GRIB files. These are is the same programs that can be invoked by the automatic ingest. The run strings for these programs are:

```
gribin setup_file GRIB_file GRIB_file...
```

```
gribin2 setup_file GRIB_file GRIB_file...
```

where ***setup_file*** is the name of your setup file and ***GRIB_file*** is the name of a GRIB file to be ingested. You may supply one or more GRIB files.

The fields from the given GRIB files will then be re-formatted for use by the FPA, and deposited in the appropriate FPA guidance directories. At this point the FPA depiction editor (**xfpa**) may be used to examine this new guidance, and the new fields may be imported.

Note

If you do not provide a absolute path to the GRIB file then it is assumed that it is in the directory pointed to by the directory tag "ingest.src". (See Directories Setup, (Section 5.5).)

3.7 Documentation

The FPA Administration Manual, Graphics Product Generator Manual, FPA Users Manual, and Graphics Metafile Standard are available in PDF format in the directory `$FPA/doc/pdf`. The following documents are available:

OpenAdminManV8.pdf	This document, formatted for double sided printing.
GPGenManV8.pdf	GPGen users manual, formatted for double sided printing.
GPGenAppendicesV8.pdf	GPGen manual appendices, formatted for single sided colour printing.
UserManV8.pdf	FPA users manual, formatted for double sided colour printing.
Metafile2.0.pdf	FPA Graphics Metafile Standard, formatted for double sided printing.

This documentation is generated using DocBook XML format. The source code can be found in `$FPA/doc/docbook` and it's subdirectories. The xsltproc tool should be easy to find using yum or apt-get and will allow you to produce documentation in html format. If you wish to produce pdf files, you can download and install dblatex. In theory xsltproc can also generate pdf files, but it does not work with the current configuration so you may have to spend some time fixing that first.

Note
You may need to adjust the include paths in your *.xsl files. Try `/usr/share/sgml/docbook/stylesheet/xsl/nwalsh/xhtml`

3.8 Third Party Software

There are several third party software packages which complement the FPA suite. These are:

3.8.1 Adobe Illustrator

Adobe Illustrator is a vector graphics editor. It is capable of reading and exporting SVG images. It can be used to edit SVGMet output or to create symbols for SVGMet products.

3.8.2 ArcGIS (with FPA Export Tool)

Note
This tool is out of date and no longer distributed with the FPA. TODO:Somebody with ArcGIS programming experience might want to try replace it.

FPA base maps and overlays can be created using the custom FPA tool developed by ESRI. See the ArcGIS to FPA Export Tool User Manual for more details.

Any map set that can be read by ArcGIS and converted to a shape file can be exported using the FPA Export tool.

3.8.3 ImageMagick

ImageMagick is a software suite that, among many other things, allows you to convert the output from PSMet to other graphics formats. It also includes the command **display**, which can be used to view these output products in what ever format they end up in.

3.8.4 Evince (evince)

PostScript products generated by PSMet (FpaGPGen in PostScript mode) can be conveniently viewed using any viewer program that can support raw PostScript format. Evince "Simply a document viewer" (**evince**) can display both PostScript (.ps) and Portable Document Format (.pdf).

Chapter 4

Contents of the FPA Directories

This chapter provides a brief overview of the contents of the FPA master directory (`$FPA`), and the structure of the data directory required in the user account(s).

4.1 FPA Master Directory

4.1.1 app-defaults

The `app-defaults` directory contains files related to the behaviour of X-windows features used by FPA applications. It contains the following directories:

Table 4.1: app-defaults directory

name	purpose
Pixmaps	Contains pixel maps used for icons in the user interface
lang	Contains X resource files for the version of the FPA which corresponds to the language as set in the `LANG` environment variable

4.1.2 bin and sbin

The `bin` directory contains the executable code for FPA applications, plus a number of support scripts. The actual executable files are found in one or more subdirectories which reflect a particular hardware and operating system. The `bin` directory contains a script which detects which sub-directory is to be used.

The `sbin` directory contains scripts to build or update FPA applications.

Table 4.2: Main executables

name	**purpose**
fpacreate_area	Allied model to create an area outline from object attributes
fpacreate_cont	Allied model to create an area outline from field contours
fpagpgen	Graphical product generator (common program)
fpaingest	Ingest scheduler
fpawarp	Allied model to adjust grid data with point data
gribin2	GRIB data ingest (Version 2 and 1 GRIB files)
gribin	GRIB data ingest (Version 1 GRIB files)
psmet	Graphical product generator (PostScript output)
svgmet	Graphical product generator (SVG output)
texmet	Graphical product generator (ASCII text output)
xfpa	Depiction sequence editor

Table 4.3: Support executables

name	**purpose**
config_check	Quality control for config files
dates	Date conversion (month and day to julian day and vice versa)
fpacfg	List referenced config files
fpadir	List all standard directories
fpaenv	List all environment variables
fpashuffle	Move files into the FPA directory structure
getmap	Define a map projection
gribtest	GRIB test decoder
hostinfo	List host name, machine ID and IP address
llmeta	Reformat metafiles in lat-lon
metadiff	Find largest differences between two FPA metafiles

Table 4.4: Master scripts

name	**purpose**
fpa.exec	Master script to select and execute the appropriate binary executable for the current hardware and operating system revision
fpapm	Controls all external operations from **xfpa**
d.fpa.exec	Backup of fpa.exec
d.fpapm	Backup of fpapm

Table 4.5: Support scripts

name	purpose
abspath	Test if path is absolute or relative to . or ..
findx	List binaries
fpabusy	Determine if a binary is being used
fpadb_minutes	Convert filenames in a directory to minutes format
fpagrep	Find a string in the FPA library code
fparev	Reports the version, revision and patch level of FPA binaries
get_config	Get path of given config file using setup references
get_directory	Get directory from setup file
get_path	Build path from directory in setup file plus filename
get_setup	Get the path of the given setup file
get_setup_parm	Get the value of the given parameter from the given setup file
pathname	Build path from directory plus filename
platform	Detect the current hardware and operating system revision and build the corresponding sub-directory name (e.g. pcLinuxvRedHat)
samehost	Determine whether two hosts area actually the same

Table 4.6: Installation/update scripts

name	purpose
copyuserlib	Install the user defined library source to a working directory
fpacleanall	Remove all compiled files from directories
fpacleandir	Remove all compiled files from a directory
fpamake	Main FPA make file
mkfpadb	Build/update an FPA database
mkfpaenv	Build/update FPA environment control file
mkfpasetup	Build setup file
pdf2fpdf	Change graphical product generator files with "pdf" extension to "fpdf" extension (Note that fpdf2pdf reverses the changes)
put_fonts	Transfer FPA fonts (in putfonts.tar) to a workstation (Note that truetypefonts_on(_off) can enable (or disable) FPA True Type fonts depending on the operating system used)

4.1.3 config

The `config` directory contains configuration files that define the elements, guidance fields, and colour tables. config files define all the possible configurations your system might have. The config directory contains the following files and directories:

Table 4.7: config directory

name	purpose
Config	Template for user configuration files
Config.***name***	Global configuration files defining all known information for FPA files and fields.
Gribs	Additional identifiers for the GRIB decoder (Note that this file is being replaced by the Ingest files listed below. It is included for backwards compatibility)
Image	Template for image configuration file
Ingest	Template for GRIB2 decoder configuration
Ingest.***name***	Global GRIB2 configuration files link GRIB2 identifiers with valid FPA sources, elements and levels.
Memory	Directory containing files which define memory presets
Menus	Directory containing files which define attribute entry menus for fields and labels
Presentation	Template for user presentation files
Presentation.***name***	Default presentation files
patterns	Directory containing line patterns
App_***name***	Specific configuration directory for each application
newconfig	Contains new versions of configuration files, from updating the FPA

4.1.4 data

The `data` directory contains data which applies to all FPA implementations. It contains the following directories:

Table 4.8: data directory

name	purpose
common	Contains common map overlays, etc.
map_dir	Contains a template database directory tree for a particular weather centre, including predefined map backgrounds and forecast area definitions
remote	One suggested destination for downloaded GRIB data from CMC Could be a NFS mount point.

4.1.5 doc

The `doc` directory contains documentation in HTML and pdf file formats.

4.1.6 etc

The `etc` directory contains the automatic ingest control file as well as templates for a number of environment control files.

4.1.7 fonts

The `fonts` directory contains print fonts used internally by the FPA.

4.1.8 ingest

The `ingest` directory is set aside for keeping track of currently active automatic ingests.

4.1.9 license

The `license` directory contains the FPA Open Source codeword file.

4.1.10 revs

The `revs` directory contains the latest list of modules that have been installed.

4.1.11 setup

The `setup` files for various FPA databases may optionally be placed in the `setup` directory. It also contains the following directories and files:

Table 4.9: setup directory

name	**purpose**
Template	Template file for building setup files using the mkfpadb script
newsetup	Receives new setup information when updating the FPA
pdf/metafiles/template	Example of a "metafile" product, for transferring FPA metafiles to another location
pdf/psmet/common	Directory for psmet predefined symbols
pdf/psmet/examples	Examples of product definition files for use with psmet
pdf/svgmet/common	Directory for svgmet predefined symbols
pdf/svgmet/examples	Examples of product definition files for use with svgmet
pdf/texmet/examples	Examples of product definition files for use with texmet
preset_lists/TEMPLATES	Examples of files for advanced interface control. The files are self documented. (See Predefined GUI Selection Lists , (Appendix F))

4.1.12 lib

The `lib` directory contains header files for FPA library code, and the shared libraries for running the FPA.

4.1.13 liblocal

The `liblocal` directory contains operational versions of the local User Defined Library (see User Defined Functions, (Appendix H)).

4.1.14 slib

The `slib` directory contains the internal libraries for running the FPA.

4.1.15 sapp

The `sapp` directory contains the libraries for building FPA applications.

4.1.16 templates

The `templates` directory contains example code for input/output of FPA data, and for developing User Defined Library code (see User Defined Functions, (Appendix H)).

4.2 FPA Data Directory

Created by the script mkfpadb (refer to Installation/update of FPA databases, (Section 3.2)).

4.2.1 Depict

The `Depict` directory contains depiction metafiles for the database.

Note
Do not add, delete or modify files in this directory while the FPA is running!

4.2.2 Backup

The `Backup` directory contains metafiles from the Depict directory that have been manually Saved.

Note
Do not add, delete or modify files in this directory while the FPA is running!

4.2.3 Interp

The `Interp` directory contains interpolated metafiles for the database.

Note
Do not add, delete or modify files in this directory while the FPA is running!

4.2.4 Guidance

The `Guidance` directory contains guidance metafiles for the database.

4.2.5 AModels.DATA

The `AModels.DATA` directory contains Allied Model metafiles for the database.

4.2.6 Maps

The `Maps` directory contains base maps and map overlays for the database and graphic products.

4.3 File Naming Conventions

There are a number of file types used to store data in an FPA database. File naming conventions for each of these file types are described below.

4.3.1 Data files

FPA data file names may not be more than 128 characters long and may not contain spaces. They must conform to one of two formats:

4.3.1.1 *eellll_YYYY:JJJ:HH[:MM]*[L]

The old format was constructed from a two letter element identifier [***ee***], followed by a level identifier [***llll***]; these were separated from the time stamp by an underscore '_'. The time stamp consisted of the four digit year [***YYYY***], three digit julian day [***JJJ***], two digit hour [***HH***], and an optional two digit minute [***MM***] separated by the ':' character. An optional 'L' character at the end indicated local time rather than GMT.

The element and level identifiers were defined in the Elements and Levels blocks of the configuration file, with the keyword 'file_id'. See Elements block, (Section A.4) and Levels block, (Section A.6).

As the number of configured elements increased the two letter element identifier was found to be too restricting. Thus, with the release of Version 7, a new naming convention was introduced.

4.3.1.2 *eeee~llll~YYYY-JJJ-HH[-MM]*[L]

The new format uses a '~' character to separate an element identifier [***eeee***], a level identifier [***llll***] and a time stamp. As before, the time stamp consists of the four digit year [***YYYY***], three digit julian day [***JJJ***], two digit hour [***HH***], and optional two digit minute [***MM***] separated by the '-' character. An optional 'L' character at the end indicates local time rather than GMT.

The element and level identifiers are defined in the Elements and Levels blocks of the configuration file with the keyword 'file_ident'. They may not contain spaces or '~' characters. The keyword 'file_id' was left alone to allow for backwards compatibility. See Elements block, (Section A.4) and Levels block, (Section A.6).

4.3.2 Map files

There are only two restrictions on map file names: They may not be more than 255 characters long; and they may not contain spaces. Map files should be located in the Maps directory defined in the setup file or they should be referred to using their full path name.

4.3.3 Image files

FPA image files are limited to 120 characters and may not contain spaces. Image files are identified using "file name masking". The mask is defined in the Image block of the Image configuration file with the keyword 'fname_mask'. See Image Block, (Section 6.3.2).

Chapter 5

Defining Your Local Setup

5.1 Customization: Setup vs. Config

Many aspects of the behaviour of FPA applications can be controlled by means of setup and configuration files.

Configuration files describe information which is relatively static. Most configuration files contain a full enumeration of available cases for a group of related parameters. Selection of desired cases is usually made in the setup files.

Setup files control the more arbitrary aspects of application behaviour. Each FPA application makes use of a setup file. When the application is started, it is provided with the name of the setup file.

The setup file must contain certain information for which no default is provided. It may also contain information defining personal preferences, which override the corresponding defaults.

Each application may concern itself with only some of this information. The information in the setup file is organized into logical blocks. The actual blocks and what they contain is to some extent arbitrary. As applications evolve new blocks may be defined or existing blocks may be altered.

5.2 Description of Setup File Contents

The setup file has a very simple format. Each block has the following structure:

```
block_name
{
  keyword value1 value2...
  keyword value1 value2...
  ... and so on
}
```

- The brackets delimiting a setup block must be on their own line.
- Lines starting with a # or * are comments.
- Quotes must be used to maintain embedded spaces.

5.3 Setup Blocks

The setup file contains the following setup blocks:

target_map	Defines the map projection information for displaying and building depictions and maps
directories	Provides the full name of important directories related to the specific FPA database referred to by this setup file
config_files	Provides the names of configuration files
interface	Defines the behaviour of the user interface for the depiction editor
depiction	Identifies the default set of fields which make up a weather depiction
ingest	Controls the behaviour of the optional automatic ingest for the FPA database referred to by this setup file
diag_control	Controls the diagnostic output for numerous modules in the FPA software
advanced_features	This block controls the behaviour of certain features in the FPA software

5.4 Target Map Setup

The **target_map** block specifies the map projection information for displaying and building depiction charts. The following key words are recognized:

Note
Latitude and Longitude values may be specified in any of the following formats.

Whole degrees	[+/-] ***DDD*** [N/S/E/W]
Decimal degrees	[+/-] ***DDD***.***DDD*** [N/S/E/W]
Degrees, minutes, (seconds)	[+/-] ***DDD***:***MM*** [:SS] [N/S/E/W]
	[+/-] ***DDD***°***MM***' [SS"] [N/S/E/W]

The default directions, with no signs present, are in degrees North for latitudes and in degrees East for longitudes.

projection

Defines the type of projection and relevant reference information.

- type of projection
- projection parameters

Projection	Type	Parameters
Lambert Conformal	**lambert_conformal**	upper reference latitude lower reference latitude
Latitude-Longitude	**latitude_longitude**	
Mercator Equatorial	**mercator_equatorial**	

Projection	Type	Parameters
Oblique Stereographic	**oblique_stereographic**	central latitude central longitude secant angle [optional]
Plate-Caree	**plate_caree**	
Polar Stereographic	**polar_stereographic**	**north** or **south** "true" latitude
Rotated Latitude-Longitude	**rotated_lat_lon**	bottom axis latitude bottom axis longitude rotation angle [optional]

mapdef
Defines the location, orientation and extent of the area of interest.

- origin latitude
- origin longitude
- reference longitude (vertical)
- x minimum (in map units given below)
- y minimum (in map units given below)
- x maximum (in map units given below)
- y maximum (in map units given below)
- metres per map unit, or degrees per map unit for latitude_longitude projections

resolution
Defines the spline resolution for calculating continuous charts.

- grid spacing (in units given below)
- metres per map unit, or degrees per map unit for latitude_longitude projections

5.5 Directories Setup

The **directories** block provides the full name of important directories related to a specific database. Each line in this block defines a directory. The first parameter is a key word used by the FPA software to identify the directory. The second parameter is the actual path of that directory.

In the default installation there will be only one entry in this block with the keyword of **home**. The software which reads this setup block will automatically change its current directory to the home directory and all relative paths will be interpreted as being relative to its path.

There are many other directory keywords, but unless there is a reason to change the directory structure from the default they do not have to be included in this block. If modification is necessary see Directories Block, (Section G.1), for the list of directories and their default values.

5.6 Configuration Files Setup

The **config_files** block defines the important configuration files used by various FPA applications. No entries are required in this block unless there is a reason to change the directory structure from the default. If modification is necessary see Configuration Files Block, (Section G.2), for the list of files and their default values.

5.7 Interface Setup

The **interface** block defines a number of customizable features in the user interface. The keywords are enclosed in square brackets and must be on a separate line from the following data. For example:

```
[map.overlay]
"FPCN20 Areas" qx_20
"FPCN21 Areas" qx_21
```

The following keywords are recognized:

allied.model
If given, specifies a list of models which may be run. Parameters are:

- model source (and sub-source if required) as ***source*:*sub-source***
- optional keyword either **notify** or **no_notify**. Specifies if the source update indicator (the book shelf) should flash when new data arrives. Default is **notify**
- optional time interval (in minutes). Any source that updates again before this time interval expires will not flash the update indicator. Default is 0.

demo.date
Set an imaginary clock if you have a "canned" database. Parameters are:

- YYYY/MM/DD/HH or YYYY/MM/DD HH

where YYYY is the four digit year, MM is the two digit month, DD is the two digit day of the month and HH is the two digit GMT hour of the day.

depiction.coview
Specifies the depictions to appear in the CoView depiction list. Parameters are:

- label to appear in the selection button
- setup file to use to view the foreign depiction
- command line options for the spawned FPA (see The Depiction Editor, (Section 2.2))
- a list of standard X options (i.e., `-geometry`, `-display`)

depiction.external
Specifies depiction databases which can be imported into or merged into the existing depiction sequence. Parameters are:

- source identifier for the required database. This must have already been defined in the configuration file as a source.
- optional keyword either **`notify`** or **`no_notify`**. Specifies if the source update indicator (the book shelf) should flash when new data arrives. Default is **`notify`**
- optional time interval (in minutes). Any source that updates again before this time interval expires will not flash the update indicator. Default is 0.

depiction.savetime
Specifies the number of days to allow saved depictions to remain in the `Backup` directory. The default is 7 days. An entry of 0 or less will allow depictions to remain in the `Backup` directory indefinitely.

depiction.timeSteps
Optional time steps for the depiction sequence, using time increments rather than the normal depiction by depiction. Parameters are:

- label to appear in the option selection list
- label to appear between the depiction time stepping arrows
- minimum time step between depictions in minutes

field.autoimport
Lists autoimport sources and specifies options. An autoimport source is a directory where valid metafiles may be put where FPA will automatically detect their presence and then import the file into the depiction sequence with actions that depend on the specified options.

- the autoimport source identifier as defined in a configuration file.
- import the field into the FPA without asking the user for permission. (**`true`** or **`false`**) Default is **`false`**.
- if importing the field would create a depiction, create the depiction without asking the user for permission. (**`true`** or **`false`**) Default is **`false`**.

field.smoothing
Specifies the maximum smoothing factor for the continuous and vector fields. This is a float value with a default of 5.0.

guidance.animationTimeSteps
Optional time steps for guidance animation frames, using time increments relative to the T0 depiction time rather than the normal frame by frame. Parameters are:

- label to appear in the selection list
- minimum time step between animation frames in minutes

guidance.model

Specifies guidance models to be made available for selection by the user. Parameters are:

- model source (and sub-source if required) as ***source*:*sub-source***
- optional keyword either **notify** or **no_notify**. Specifies if the source update indicator (the book shelf) should flash when new data arrives. Default is **notify**
- optional time interval (in minutes). Any source that updates again before this time interval expires will not flash the update indicator. Default is 0.

imagery.blend

Set the lower limit of the radar-satellite image blending amount in percent. For example, a value of 40 will limit the blending to 40% radar and 60% satellite. The upper limit is always 100% radar meaning that the radar blocks out all of the satellite.

- minimum amount of radar blended in percent.

imagery.brightness

Sets the range of brightness adjustment, in percent, allowed for the various image types.

- One of **overlay**, **radar**, **satellite** or **underlay**.
- Minimum brightness. No less than 0%.
- Maximum brightness. Can exceed 100% which will make the image brighter than the original source.

For example:

```
[imagery.brightness]
overlay     30  100
radar       50  100
satellite   50  100
underlay    30  100
```

interpolation.delta

Specifies the time step between interpolated depictions in hours, or in hours:minutes. The default is 1 hour.

map.base

Specifies the main background map. Parameters are:

- label to appear in the background selection button
- metafile name (relative to `Maps` directory)

There must be one entry.

Note: A '+' prefixed to the map metafile name indicates a map which is found in the "maps.common" directory as opposed to the "maps" directory.

map.overlay
Specifies the available map overlays. There is no limit as to the number of overlays allowed. Parameters are:

- label to appear in the background selection button
- metafile name (relative to `Maps` directory)

Note: A '+' prefixed to the map metafile name indicates a map which is found in the "maps.common" directory as opposed to the "maps" directory.

map.editor
Specifies a map containing an outline used in the field editor. Parameters are:

- label to appear in the editor selection button
- metafile name (relative to "maps" directory)

Note: A '+' prefixed to the map metafile name indicates a map which is found in the "maps.common" directory as opposed to the "maps" directory.

map.holes
Specifies a map containing an outline used in the field editor for drawing holes in area fields. Parameters are:

- label to appear in the editor selection button
- metafile name (relative to `Maps` directory)

Note: A '+' prefixed to the map metafile name indicates a map which is found in the "maps.common" directory as opposed to the "maps" directory.

map.palette
Specifies the available map colouring schemes. All colours are to be given as X colour names or in X hexadecimal notation. A dash ("-") in place of any given colour means to use the default. There must be at least one entry. Parameters are:

- label to appear in the map palette selection button
- land colour
- water colour
- coast colour
- border colour
- latitude/longitude colour
- forecast area colour
- forecast area border colour

link.palette
Lists the available link chain colouring schemes. All colours must be given in standard X colour names. A dash ("-") in place of any given colour means to use the default. Parameters are:

- label to appear in the link palette selection button

- colour for link nodes and chains
- colour for labels
- colour of link node that is not attached to a feature

product.graphic
Specifies graphic output products. Note that the format of product definition files is discussed in Product Definition File Formats, (Appendix D). Parameters are:

- label to appear in the graphic product list
- type of graphic product one of:

psmet	produce graphics using PSMet
svgmet	produce graphics using SVGMet
texmet	produce text file using TexMet
metafiles	package up the FPA database for transmission to another location

- parameters specific to type:

  psmet, svgmet, texmet

  - directory relative to "psmet", "svgmet" or "texmet" setup directory
  - product definition file in the above directory

  metafiles

  - product definition file. If an absolute path is not specified, then this is taken relative to the "metafiles" setup directory.

title
If given, specifies a title to be used in the program window. The default is the name of the FPA database.

T0.roundOff
Set the time roundoff for setting T0 in hours. This will roundoff T0 to the nearest time boundary defined by the setting. For example, 6 will result in a T0 to the nearest 6 hour interval. If the current time was 16:36 GMT then T0 would be set to 12 GMT.

5.8 Depiction Setup

The depiction block defines the list of fields that make up a depiction and may be edited using the depiction editor application (**xfpa**).

field
Identifies a field in the depiction. Parameters are:

- element name
- level name

Elements and levels mentioned in this block must already be defined in the Config file (refer to Configuration Files, (Chapter 6)).

5.9 Ingest Setup

The ingest block controls the ingest of CMC model output (GRIB files). The following keywords are recognized:

wait
Defines the time (in seconds) that the ingest Daemon will go to sleep before checking for new data again. If this line is omitted, a default (300) will be used

log
Identifies the log file name and backup log file name to be used. At a specified time each day, the log file is renamed to the backup file name and a new log is started. This prevents unreasonable growth. Parameters are:

- log file name (default `ingest.log`)
- backup log file name (default `ingest.old`)
- backup time (GMT) (default 0000)

The log file and backup log file will be placed in the 'ingest.log' directory, as defined in the directories block, unless an absolute path is given.

status
Identifies the status file to be used (default `ingest.stat`). The status file will be placed in the 'ingest.stat' directory, as defined in the directories block, unless an absolute path is given.

monitor_in
Identifies a set of data files to be monitored by the Ingest Daemon. Files are assumed to be in the 'ingest.src' directory, as defined in the directories block, unless an absolute path is given. File names here can use the **sed(1)** style regular expressions. If one or more of the specified files arrives, its new modification time is detected, and the ingest program is started up to process it. Parameters are:

- directory tag (usually **ingest.src**)
- type (**grib** or **grib2**) or application (gribin, gribin2 or your own ingest program)
- pattern for matching file names

Note
You may use one of the included ingest programs, or supply your own ingest program. The command line parameters of your program must match those of the included ingest programs:

```
ingest_prog setup_file data_file [ data_file ... ]
```

5.10 Diagnostic Control Setup

This block controls the diagnostic output from numerous modules if the FPA software.

Behaviour of diagnostic output may also be determined or altered by external influences, such as run-string arguments. E.g. the `-debug` argument in **xfpa** causes the information in this block to be used, otherwise a default level is used and this block is ignored.

The following key words are recognized:

default
defines the default output options for all modules

- level
- message

module
defines the output options for the given module

- Module name
- level

Table 5.1: Output levels

level	meaning
0	No output
1	Errors only
2	Include warnings
3	Include status messages
4	Include information messages
5	Include diagnostic messages

Table 5.2: Message styles

level	meaning
0	Module name does not appear with message
1	Module name followed by ':'
2	Module name in '[ ]'

Module names are completely arbitrary, and already exist within the FPA library code. It is necessary to know about given modules in advance, in order to make effective use of this block.

Table 5.3: Module names accessible by users

module	**meaning**
Editor	General FPA interface
Editor.Events	FPA window
Editor.Feedback	from FPA libraries to GUI
Editor.API	from GUI to FPA libraries
LogMsg	general FPA log messages
Config	Config files
Environ	files and directories
Fields	depiction fields
Metafile	reading & writing metafiles
Presentation	Presentation files
Show.Patches	show the FPA 'grid' based on Bspline patches
Advanced.Features	Advanced Features block
Diag.Control	Diagnostic Control block
Limits	specialized limit boxes
Contouring	problems with contouring
MMM	memory management
Patch.Control	spline patch tracking
Tracker	contour tracking
Area	problems with area objects
Area.Divide	problems with dividing lines
Line	problems with line objects
Line.Walk	distances along boundaries or lines
Sculpt	sculpting areas or lines
Spots	problems with spot objects
Timelink	related to time linking
Interp	general interpolation
Interp.Areas	interpolating area objects
Interp.Curves	interpolating line objects
Image	imagery
Rules	attribute rules
FPA_mod100_spval	Evaluate last 2 digits of pressure rule
FPA_wx_label	Evaluate weather label rule
FPA_wx_category	Evaluate weather category rule
FPA_clds_and_wx	Evaluate clouds and weather rule
FPA_full_weather	Evaluate full weather rule
user_clds_and_wx	Evaluate user-defined clouds and weather rule
user_full_weather	Evaluate user-defined full weather rule
user_full_cloud	Evaluate user-defined full cloud rule

5.11 Advanced Features Setup

This block controls the behaviour of certain advanced features in the FPA software.

feature
defines the module to be used for a given feature. Parameters are:

- feature name
- mode

Feature names are completely arbitrary, and already exist within the FPA library code. It is necessary to know about given features in advance, in order to make effective use of this block.

5.12 Resource File `.XFpa`

A template of this file is located in `$FPA/etc/d.XFpa`. To override resource settings for FPA copy this file to your home directory, rename it to `.XFpa` and customize it.

```
cp $FPA/etc/d.XFpa $HOME/.XFpa
vi .XFpa
```

This file is useful if you wish to override any display characteristics of the depiction editor. The file is internally documented. The following is a listing of some of the more useful characteristics that can be set using the `.XFpa` file.

xfpa.confirmExit
The usual exit from FPA brings up a confirmation window. The window can be bypassed by setting this to **`no`**. Default is **`yes`**

xfpa.sequenceIncrements
The usual behaviour of the forward/backward arrows to the right of the time sequence is to step through the depictions one by one, regardless of the time between depictions. It is possible to allow for fixed time steps here. This list consists of groups of three comma separated items:

- the label to appear in the selection list,
- the label to appear between the arrows,
- and the minimum time step between depictions in minutes.

these three parameters may be repeated to add several fixed time step options.

xfpa.sourceUpdating.FlashTime
Set the length of time, (in minutes) after a source has finished updating, that the indicator light will flash.

xfpa.sourceUpdating.IndicatorDelay
Set the length of time, (in minutes) after the user has checked the source status with the ***Guidance Status ...*** dialog, that the indicator light will not flash even if the source is updating.

Chapter 6

Configuration Files

Most FPA applications make use of information that is provided in a set of configuration files. Unlike information in the setup files, this information is relatively static. Configuration files usually enumerate all possible values of a given parameter, or provide a default. Setup files select a preferred value or override the default. It is not necessary to keep different copies of configuration files to reflect different preferences. The existing configuration files should reflect the office preference only. The following configuration files are used:

Config, Config.***name***	Provides all information about fields, elements and levels
Gribs, Gribs.***name***	Provides mapping for unknown GRIB identifiers (**gribin**) Note that this file is being replaced by the `Ingest` files listed below. It is included for backwards compatibility.
Image, Image.***name***	Provides information about satellite and radar imagery
Ingest, Ingest.***name***	Provides mapping for unknown GRIB identifiers (**gribin2**)
Presentation, Presentation.***name***	Defines the default appearance of all known fields when displayed by graphical FPA applications.
Memory/*	A series of files which define memory presets
Menus/*	A series of files which define the contents of attribute entry menus for fields and labels

The '.***name***' notation is used to indicate a series of files beginning with the same name but ending with different suffixes to indicate files containing specialized versions of configuration information.

6.1 Config and Config.*name*

The `Config` and `Config.`***name*** configuration files define information about all files and fields recognized by the FPA. The `Config.`***name*** files are global configuration files containing standard FPA information as well as information for specific FPA applications. The `Config` file is a local configuration file which will include one or more of the `Config.`***name*** files, as well as contain user-defined changes or additions to these files.

A complete description of the format and contents of Config files is given in Config and Config.***name*** File Format, (Appendix A). Information in the Config files is arranged in blocks. The primary blocks are:

Table 6.1: Primary blocks of config files.

name	**usage**
Elements	any meteorological parameter
Levels	levels of the atmosphere
Fields	an element at a specific level
Groups	grouping for lists of Fields or Elements
CrossRefs	identifying special methods to calculate Values or Winds
Samples	identifying special methods to sample Values or Winds
Sources	sources of input or output data
Constants	meteorological values used in equations
Units	units and conversions to MKS

The `Config` file may contain as many blocks as desired, and blocks can be repeated if required. Each block contains a number of members, and information for those members is identified by keywords. The first reference to a member of a block controls the order of that member in any lists for that block, and the last reference to a member controls the values for any given keyword. Placing information later in a `Config` file therefore overrides any information that has been identified earlier.

The general format of the `Config` files is:

```
revision revision_number
include filename
include filename
block_name
  {
  member_name
    {
    keyword = value(s)
      OR
    keyword =
      {
      keyword = value(s)
      }
    }
  }
```

In some cases, there are also ***sub_block_name*** or ***sub_member_name*** sections, but this general format is seen throughout. Config and Config.***name*** File Format, (Appendix A) identifies the recognized keywords for all Config file information.

Note
that the code for reading the Config files is fairly simplistic, and that the format is important. For example,

- all '{' or '}' characters must be located on their own lines,
- the '=' must be separated from the rest of the line by white space, and
- line continuations '\' must be used if the ***keyword*** = ***value(s)*** format will not fit on one line.

6.2 Presentation and Presentation.*name*

The `Presentation` and `Presentation.`***name*** files define information about how to display fields that FPA could potentially encounter. The `Presentation.`***name*** files are global presentation files containing standard display information as well as information for specific FPA applications. The `Presentation` file is a "local" presentation file which will "include" one or more of the `Presentation.`***name*** files, as well as contain user-defined changes or additions to these files.

6.2.1 Presentation Format Summary

A complete description of the format and content of Presentation files is given in Presentation and Presentation.***name***, (Appendix C). The entries in a Presentation file are grouped by field blocks. The following represents the basic format of a presentation file entry.

Note
When a parameter value must come from a predefined list, this list will be presented between '{' and '}' and the options are separated by the '|' character. Optional attributes are bracketed between the '[' and ']' characters. The "appropriate display options" mentioned in the summary below specify how the objects are to be displayed. Since many of the display options are common amongst the different member types they are described once at the end of Presentation and Presentation.***name***, (Appendix C).

```
revision revision_number
field element level source
  member {continuous | vector} dname
    units uname
    contour range min max standard increment
      appropriate display options
    contour list val val ...
      appropriate display options
    vector multiplier (vector only)
      appropriate display options
    maxima min max
      appropriate display options
    minima min max
      appropriate display options
```

```
    saddle min max
      appropriate display options
  member {nodes | spot} dname
    class cname
      class_member {barb | button | label | mark} mname attribute
        category value
          appropriate display options
        attribute name value
          appropriate display options
        default
          appropriate display options
  member {area| barb| button| curve| discrete| label| lchain| mark} dname
    category value
      appropriate display options
    attribute name value
      appropriate display options
    default
      appropriate display options
  member plot dname
    subfield {barb | button | label | mark} sname
      appropriate display options
include include_file
```

Most blocks (shown by indentation here) can be repeated as many times as needed. However, for some blocks (vector, maxima, minima, saddle, default) it doesn't make sense to have more than one per member or class_member block.

6.3 Image

The `Image.`***`name`*** files define how rasterized images are displayed in the FPA editor. Images are associated with product groups with a common type (radar or satellite) and appearance (colour tables).

In the `Image` file, two types of data are specified. Each must be in a block of information in the following format:

```
block_key block_tag
  {
  information
  ...
  }
```

The ***`block_key`*** is either **`product`** or **`image`**. The ***`block_tag`*** is whatever you want that will make the file easy to read. Images are grouped into products. Thus product blocks stand on their own, but an image block must belong to a product block.

6.3.1 Product Block

A product is a group of images to which certain attributes apply. The product is identified by a ***`product_tag`*** which is then referred to in the image entry block. The product block format is:

```
product product_tag
  {
  key_word = item list ...
  key_word = item list ...
  ...
  }
```

Key words are:

label
The product label as seen by the end user

class
The type of image the group is for. One of: **radar**, **satellite**, **overlay**, **underlay** or **geographic**.

The "overlay" is an image that will appear over the satellite and radar, while "underlay" will appear under satellite and radar.

Geographic images do not have a time component and are meant for static images such as geography or political boundaries.

ctable
Either the location of a colour look up table file to be applied to images in the product group or the special key 'DefinedByImage'.

The key 'DefinedByImage' means that the images associated with the product must define their own colour tables. This means that they can not be changed as there will not be just one colour table for all of the images included in the product. (See the entry for ctable in the image block section).

There can be more than one colour table defined and they are expected to be selectable by the user.

There are 4 parameters:

- The label to give to the look up table.
- The directory key, from the setup file, where ctable file is located.
- The name of file containing the ctable.
- Optional parameter **default**. If specified the given ctable is set as the default.

Note
Look up tables come in two forms, those which assign a particular pixel value to a colour and those which are used with data files to assign a range of values (reflectivity, cloud top temperature, etc) to a colour.

Note
If there is no lut entry in the product block or the ***directory_key*** and ***file_name*** are either missing or set to a dash ("-") then the display will be the one embedded in the image itself (like GIF).

6.3.2 Image Block

A particular image is identified by an ***image_tag*** and a valid time. The `Image` file provides other necessary information for processing the image, such as its location and filename, as well as geo-referencing information.

Note
The order of the images in the configuration file will determine the display order as seen on the screen. The first image in the file will be displayed on top of the set of images. The second will be displayed "underneath" the first and on top of the third and so on.

The Image block format is:

```
image image_tag
 {
 key_word = item list
 key_word = item list
 ...
 }
```

Key words are:

site or label
The name of the radar site (e.g. "King") or of the satellite that the image comes from (e.g. "GOES West")

product_tag
The product group to which this image belongs. See product_tag above.

ctable
If the product group ctable entry is 'DefinedByImage' then any images that require a colour table must set it here.

Note
Unlike the product entry only one ctable can be defined.

There are two parameters:

- Directory key from setup file.
- Name of file to read.

encode
How the image is encoded. The recognized encoding types are listed in Table 6.2.

Table 6.2: Image Encoding

type	meaning
none	The image is an unencoded one byte per pixel grey scale image
rgb	The image is an unencoded three bytes per pixel colour image stored as rgb triples (rgbrgbrgb...)
any	The library attempts to determine the image format from the file extension and file contents.
gif	Stored as a GIF file
tiff	Tag Image File Format.
png	Portable Network Graphics file.
xwd	X window dump format
xgl	The format native to this library.
gridded_data	Data is an unencoded array of scaled data values stored as value = pixel * scale + offset. See below.
urp_gridded	Stored as universal radar processor data in gridded format.
urp_polar	Stored as URP format in range-theta format.
FpaMetafile	The image or raster of scaled values is in an FPA metafile. The actual determination as to which type is done from the meta data in the file.

force_grayscale
: **`true`** or **`false`** (default). If set the image is forced to grey scale. This is provided in case the image (especially satellite) had a colour label added. This would cause the entire image to be treated as a colour image, which is not usually wanted.

aspect_ratio
: **`adjust`** or **`fixed`** (default). The aspect ratio of the image is allowed to change depending on the view port setting or must remained fixed.

For all image files (such as gif, png or tiff)

transparent
: Set the given colour to transparent. There are two parameters:

- Either the name of a colour or the red, green and blue values of the colour to be taken as transparent.
- Optional parameter "closest". The optional key is only used for image files with a 256 element colour table, like GIF, and will use the colour closest to the requested transparent colour. Useful in cases when colours like white are "sort of" white. (example: transparent = white closest, transparent = 201 76 158).

More than one transparent entry is allowed and all of the designated colours will be taken as transparent.

print_cmaps
: For images with 256 element colour tables this will print the tables if in diagnostic mode. For temporary use to see what colours are used in a image. There is one parameter:

- True or False.

Default False.

For all images except geographic types:

directory
Specifies the location of the `Image` files. There are two parameters:

- The base directory key name of the image directories as found in the setup file.
- Directory where the images are to be found.

fname_mask
The mask which is used to read the list of files from the above directory.

The following will return all files starting with any characters in the first 12 positions followed by the string _WSO_CAPPI (and possibly followed by other unspecified characters).

```
fname_mask = ............_WSO_CAPPI
```

Note that the mask uses regular expression matching, so that "." matches any single character, "*" matches any string of characters, and "$" indicates the end of the name.

Thus the file `200005132130_WSO_CAPPI_RAIN.GIF` would be a match for the previous example, but not for:

```
fname_mask = *_WSO_CAPPI$
```

If not specified the default is all files ("*"). Filenames cannot be longer than 120 characters.

fname_time
The parsing format for the time contained in the file name in the form of ***format parse_keys***. Format is the way to parse the string in the standard c sscanf style notation and parse_keys give the meaning of each of the elements parsed. In the above example the file starts with the date, so the entry **`%4d%2d%2d%2d%2d YYYY MM DD hh mm`** will parse it. If there was a string in front of the date this would need to be included in the format string. For example if we had:

```
fname_mask = RT............_WSO_CAPPI
```

then the format would be:

```
fname_time = RT%4d%2d%2d%2d%2d YYYY MM DD hh mm
```

The default is "**`%4d%2d%2d%2d%2d YYYY MM DD hh mm`**". The parse keys are case sensitive. The recognized parse keys are listed in Table 6.3.

Table 6.3: Parse Keys

key	meaning
YYYY	4 digit year
YY	2 digit year (the last 2 digits of the year)
JJJ	3 digit julian day

Table 6.3: (continued)

key	meaning
MM	2 digit month
DD	2 digit day of month
hh	2 digit hour of the day
mm	2 digit minute of the hour

time_frequency

The time between images. If no time information is to be specified (or if the interval time cannot be specified as is the case for polar orbiting satellites), then enter "none" to avoid complaints from the configuration reader.

The time frequency information consists of four entries:

- the start time as an offset from 00 GMT
- the cycle time
- acceptance window before.
- acceptance window after.

All time entries can be specified in minutes (m) or hours and minutes in the format h:m (for example 15 or 1:10). The acceptance window is a time within which an image can arrive and still be considered to be on time. Some satellite images that normally arrive on the half hour may actually arrive at times like 10:32, thus an acceptance window of 2 minutes would consider this to be the 10:30 image. Typical entries might be:

```
radar 0:0 10 0 0
satellite 0:0 15 2 5
```

If a satellite repeated every half hour starting on the quarter hour, plus or minus 2 minutes, the entry would be:

```
time_frequency = 15 30 2 2
```

The following are required for radar and satellite images only (like GIF) and satellite raw raster data files. If projection and mapdef are specified for URP or TDF files, then the entries in the `Image` file will override the information in the data file header. If not used, mapdef should be specified as "none" or the configuration reader will complain.

projection

Defines the type of projection and relevant reference information.

- type of projection
- projection parameters

Projection	Type	Parameters
Lambert Conformal	**lambert_conformal**	upper reference latitude lower reference latitude
Latitude-Longitude	**latitude_longitude**	
Mercator Equatorial	**mercator_equatorial**	
Oblique Stereographic	**oblique_stereographic**	central latitude central longitude secant angle [optional]
Plate-Caree	**plate_caree**	
Polar Stereographic	**polar_stereographic**	**north** or **south** "true" latitude
Rotated Latitude-Longitude	**rotated_lat_lon**	bottom axis latitude bottom axis longitude rotation angle [optional]

mapdef
The map definition of the image. FPA mapdefs are described in the FPA Graphics Metafile Standard. Entries are:

- origin latitude
- origin longitude
- reference longitude (vertical)
- x minimum (in map units given below)
- y minimum (in map units given below)
- x maximum (in map units given below)
- y maximum (in map units given below)
- metres per map unit, or degrees per map unit for latitude_longitude projections

Note
For radar numeric files the mapdef can be set to none and the appropriate map definition information will be read from the data files.

Key words for radar image files only (like GIF):

radar_overlay
If the image has the geography in specific planes in the image this can be used to remove them. Parameters are:

- either **on** or **off**
- an 8-bit mask indicating which bits in the image are to be used or rejected. For example 00111111

radar_bgnd
Do we want the image background to be **transparent** or **opaque**?

radar_bg_color
The rgb (red, green, blue) values of the colour to be taken as transparent if radar_bgnd is set to **`transparent`**. The values must range from 0 to 255 and the default is black (radar_bg_color = 0 0 0)

radar_extent
Display the entire image (**`full`**) or just the part where the radar scan is (**`data`**)?

radar_centre
Where in the image the centre of the radar scan can be found. Entries are width and height in pixels

radar_diameter
The diameter of the radar scan part of the image in pixels

range_rings
If **`true`** the library is allowed to overlay range rings on the image. Many radar images have range rings already in the image so normally this key word is either left out or set to **`false`**

For satellite images only:

mapdef_file
Some images, such as those received from polar orbiting satellites have a map definition that changes from one image to another. This entry defines the name of the file that contains the map definition and/or projection. There are two forms the files can take:

1. The file contains the definitions for one image and the file name identifies which image to information is for. The file name is in the form of ***`format encode_keys`***. Format is the way to encode the file name in standard C sprintf style notation and encode_keys give the meaning of each of the elements. For example: %2.2d%2.2d%2.2d%2.2d%2.2d_WSO_CAPPI.md ef YY MM DD hh mm where the time is encoded according to the rules defined above for the fname_time. the file must contain the definitions in the format given in the discussion of the projection and mapdef above although the equal ("=") sign is not required.
2. The file contains the definitions and/or projections for more than one image. In this case the item list contains only one entry which is the name of the file. For example: `mapdefs` Each line in the given file must be in the form of:

   ```
   image_file_name key parameters
   ```

   where ***`image_file_name`*** is the name of the image file, ***`key`*** is one of **`mapdef`** or **`projection`**, and the ***`parameters`*** are as described in the discussion above.

The files are assumed to be in the same directory as the images. Note that the map definition and projection must still be specified in the `Image` file for use as the default if the file is not found or if only the map definition is specified in the file.

Key words for the satellite or data with encode key **`gridded_data`** only:

element
The name of the element encoded into the grid (e.g. temperature)

size
The size of the image grid. Entries are width and height in pixels.

bytes_per_pixel
The number of bytes (8-bit chunks) per pixel (must be one of 1, 2, or 3)

byte_order
Either **`MSBFirst`** for Most Significant Byte First or **`LSBFirst`** for Least Significant Byte First. Only valid for bytes_per_pixel >1.

scale and offset
Both scale and offset are used to turn the pixels into values represented by the pixels, and are applied as pixel * scale + offset For example if cloud top temperature was encoded as degrees Kelvin*100 and we wanted degrees Celsius then the scale and offset values would be

```
scale = 0.01
offset = -273.15
```

Key words for encode key **`none`** only:

size
The image dimensions, width and height in pixels

For geographic images only:

file

- The base directory key name of the image directories as found in the setup file, or an absolute pathname of the directory (starts with the character '/').
- The full path name of the file starting from the base directory.

6.3.3 XGL image file format

This is the native format of this library. It is a very simple uncompressed raster format and consists of 24byte header block followed by a byte that is the type of image storage. The options are G, R, P and mean:

- G = 1 byte per pixel (grey-scale)
- R = 3 bytes per pixel stored as RGB triplets: RGBRGBRGB...
- P = 3 bytes perp pixel stored as RGB in plane order: RRRR...GGGG...BBBB

This is followed by a 14 byte block that holds the image size stored as width x height. Note that the number of bytes in the image will be width*height*bytes-per-pixel. Thus the file size (bytes) will be 24+ width*height*bytes-per-pixel. Header example: "xglrasterR1024x768 "

6.4 Gribs

Warning
The `Gribs` file is obsolete, it is being replaced by the `Ingest` file (see Ingest and Ingest.***name***, (Section 6.5)). It is included for backwards compatibility.

The `Gribs` file describes model identifiers or element identifiers not yet recognized by the FPA GRIB decoder **gribin** (For GRIB editions 0 and 1). The following keywords are recognized:

gribmodel
Defines GRIB model identifiers not presently recognized by the FPA GRIB decoder **gribin**. The parameters are:

- GRIB forecast centre identifier number
- GRIB forecast centre model identifier number
- FPA model label (must be a valid source in Config)

gribelement
Defines GRIB element identifiers not presently recognized by the FPA GRIB decoder **gribin**. The parameters are:

- GRIB element identifier number.
- FPA element label (must be a valid element in Config)
- FPA units (must be valid units in Config).

gribfieldskip
Defines fields which will not be processed by the FPA GRIB decoder **gribin**. The parameters are:

- FPA element label (must be a valid element in Config) or can be **ANY**
- FPA level label (must be a valid level in Config) or can be **ANY**.

6.5 Ingest and Ingest.*name*

The `Ingest` files map model, element and level information from an input format, such as GRIB, into model, element and level tags recognized by the FPA. Currently GRIB editions 0, 1 and 2 are supported by the ingest program **gribin2**. The `Ingest.master` file contains mappings for common models, elements and levels. The `Ingest` file is for local customization and should include the `Ingest.master` file. The `Ingest` file format is:

```
revision revision_number
include file_name
include file_name
Gribs edition
  {
```

```
  block_name
    {
    ...
    }
  ...
}
```

The revision line must be the first uncommented line in each ingest configuration file. The include lines allow several ingest configuration files to be used at the same time. The ingest configuration file reader merges the information from the included files with the information in the current file to determine all the rules needed to ingest the GRIB files successfully. When a query matches more than one rule, the last rule read by the ingest configuration file reader will be used.

The general format does not allow line concatenation; for example,

```
block_name {
```

on the same line would not be recognized.

The Gribs ***edition*** is one of {0,1,2}, and ***block_name*** is one of:

Sources	A set of rules to assign FPA source names to a GRIB model
Elements	A set of rules to assign an FPA element name to a GRIB element
Levels	A set of rules to assign an FPA level tag and scale information
Fields	A list of fields to either skip or process into FPA Metafiles
DataFiles	A list of fields to either skip or process into a raw Datafiles that can be viewed as pseudo-satellite images in xfpa.

6.5.1 Sources block

The **Sources** block sets rules to map GRIB model identifiers to valid FPA sources. The general format of the **Sources** block is:

```
Sources
  {
  source_name
    {
    parameter = p0 p1 ... pn
    parameter = p0 p1 ... pn
    ...
    }
  ...
  }
```

where ***source_name*** is the source tag to be assigned, and ***p0*** through ***pn*** are identifiers determined by the GRIB edition.

editions 0 & 1

- p0 = Originating centre identification number (centre)

- p1 = Model identification number (model)

edition 2

- p0 = Originating centre identification number (centre)
- p1 = Sub centre identification number (sub_centre)
- p2 = Product template number (template)
- p3 = Type of forecast product (forecast_type)
- p4 = Process identification number (process)
- p5 = Background process identification number (bkgd_process)

The '*' character can be used to specify all of something. For example, if you want all models coming from centre 54 to have the source tag GEM then the following parameter values could be used:

```
parameters = 54 *           # for GRIB editions 0 and 1, or
parameters = 54 * * * * *   # for GRIB edition 2.
```

Some restrictions apply to the use of '*' in the **Sources** block. Generally once you use a '*' all parameters that follow on that line must also be '*'. There is one exception to this rule; in GRIB edition 2, you may set sub_centre to '*' and still specify forecast_type, template, process and bkgd_process. If you want a range of model ids to match a single source tag, you can specify as many parameter lines as you need.

When two or more rules match a query, the more specific rule will be applied. If there is a tie the "newest" rule will apply. Thus it is important to make your customizations after the `Ingest.master` file is included. Source names should be previously defined in the `Config` file to be recognized.

6.5.2 Elements Block

The **Elements** block sets rules to assign FPA element and unit tags to GRIB fields. Each GRIB producer has the option to define their own element parameters within the ranges reserved for local use. The source list is used to identify an element parametrization with a particular source or list of sources. If the parametrization is from the standard range then the source can be set to **default**. These parameters will be valid for all sources.

```
Elements
  {
  source_names
    {
      element units
        {
        parameters = p0 p1 ... pn
        parameters = p0 p1 ... pn
        ...
        }
    }
  }
```

where ***source_names*** can be a single source, a list of sources or **default**, and ***p0*** through ***pn*** are identifiers determined by the GRIB edition

editions 0 & 1

- p0 = Version number of the parameter table (table_version)
- p1 = Parameter id (parameter)

edition 2

- p0 = GRIB Master Table number (discipline)
- p1 = Product definition template number (template)
- p2 = Parameter category (category)
- p3 = Parameter identification number (parameter)

The '*' character can be used to specify "all" of something. In the **Elements** block the '*' character is only acceptable in place of template or table_version. Element and unit names should be previously defined in the `Config` file to be recognized.

6.5.3 Levels Block

The **Levels** block sets rules to assign FPA level tags to GRIB levels. Each GRIB producer has the option to define their own levels within the range reserved for local use. The source list is used to identify a level id with a particular source or list of sources. If the level id is from the standard range then the source can be set to **default**. These levels will be valid for all sources.

```
Levels
  {
  source_names
    {
    level_name
      {
      parameters = p0 p1 ... pn
      parameters = p0 p1 ... pn
      ...
      }
    }
  }
```

where ***source_names*** can be a single source, a list of sources or **default**, and ***p0*** through ***pn*** are identifiers determined by the GRIB edition

editions 0 & 1

- p0 = Level type: surface=**0**, level=**1** or layer=**2** (level_type)
- p1 = GRIB level identification number (level_id)
- p2 = Amount to scale level (or first level of a layer) value by (scale_1)

- p3 = Amount to offset level (or first level of a layer) value by (offset_1)
- p4 = Amount to scale upper level of a layer by (scale_2)
- p5 = Amount to offset upper level of a layer by (offset_2)

edition 2

- p0 = Level type: surface=**0**, level or layer=**1**(level_type)
- p1 = GRIB level identification number (level_id)
- p2 = Amount to scale level by (scale)
- p3 = Amount to offset level by (offset)

In both edition 1 & 2, the scale and offset values are used to change the units of the level or layer values to MKS or something more convenient for FPA. FPA uses level/layer values to specify a level tag. The values need to be integers for this purpose. Level names should be previously defined in the `Config` file to be recognized.

6.5.4 Fields Block

The **`Fields`** block is used to specify a list of fields to process or skip when generating FPA metafiles.

```
Fields
  {
  process  = source element level
  skip     = source element level
  redirect = original destination
  }
```

command	purpose
process	Convert only selected fields into FPA metafiles
skip	Convert all but the selected fields into FPA metafiles. You cannot combine process and skip directives. If a process directive exists then all skip directives are ignored.
redirect	Redirect from origin source to destination source for processed fields. This allows us to separate metafiles and datafiles generated from the same GRIB field into different directories. Here the destination source must be defined in a `Config` file.

The '*' character may be used to match all possible choices for source, element or level.

```
process = GFS   *   surface
```

will process all surface fields from the GFS source.

6.5.5 DataFiles Block

The **`Datafiles`** block specifies a list of fields to process or skip when generating FPA datafiles.

```
DataFiles
  {
  process  = source element level
  skip     = source element level
  redirect = original destination
  rescale  = element level scale offset
  }
```

command	purpose
process	Convert only selected fields into FPA data files
skip	Convert all but the selected fields into FPA data files. You cannot combine process and skip directives. If a process directive exists then all skip directives are ignored.
redirect	Redirect from origin source to destination source for processed fields. This allows us to separate metafiles and datafiles generated from the same GRIB field into different directories. Here the destination source must be defined in a `Config` file.
rescale	A datafile is a binary file which contains a simple grid of data. Each datum is stored as a signed short integer (2 bytes). If any of the data being stored falls outside the range of signed shorts, information will be lost. Rescale the data by scale and offset: **`short_val = (short int) ((float_val - offset)*scale)`** to avoid losing information. The '*' character may be used to match all possible choices of element or level. Scale and offset may be floating point numbers. example: rescale = temperature * 0.1 0.0

The '*' character may be used to match all possible choices for source, element or level.

```
process = GFS * surface
```

will process all surface fields from the GFS source.

6.6 Memory Preset Configuration File Format

The files in the Memory directory contain the definitions of predefined lists of editor memory items. These items will always be placed before those defined by the users. Each file is associated with a particular field and is specified in the `Config` file by the **`memory_file`** or **`background_memory_file`** keyword in the **`editor`** section of the **`Elements`** block.

Each block of data must be proceeded by the keyword 'memory'. After this are attribute - value pairs which define the memory contents. Any attribute not listed will be assigned its default value. There are three special attributes which can be used by many fields.

attribute	purpose
FPA_user_label	the user set label for the entry
FPA_auto_label	the 'value' of the area
FPA_category	the 'category' of the area

```
memory
 FPA_user_label = Showers
 cloud_base     = 25
 precip         = RW-
 vis            = 1-3
```

Note
The actual category names will vary according to what is defined in your configuration files.

6.7 Attribute Entry Menu Files

The files in the Menus directory contain definitions of attribute entry menus. The name of the file is set in the **Elements** block of the Config file. There are two types of entry files, Attribute Entry Menu files and Wind Entry Menu files. (Wind Entry Menu files are described in the following section.)

The attribute entry menu configuration files determine the appearance and content of the attribute entry menus. Each file is associated with a particular field or label and is specified in the Config file by the **background_entry_file**, **type_entry_file**, **type_modify_file**, **entry_file** or **modify_file** keywords in the **editor** section of the **Elements** block. (Config and Config.***name*** File Format, (Appendix A)). These files are located in the Menus configuration directory.

The first six lines of the attribute entry menu files define values which apply to the preset controls in the menu. These are:

Table 6.4: Attribute Entry Menu - Preset Controls

control	usage
menu_title	Sets the title to appear in the menu frame at the very top of the window.
geometry	Sets the size of the entire entry menu as x y width height all given in pixels. The position x y is relative to the button which causes the menu to appear and can thus have a negative value.
label_display_height	Set the height, in rows, of the User Label display. If set to zero (0), then the User Label display will not appear in the menu. The default is 1
value_display_height	sets the height, in rows, of the Auto Label display. If set to zero (0) then the Auto Label display will not appear in the menu. The default is 2
category_display_width	Sets the width of the category display in characters the category display is located just to the right of the Auto Label display. If set to zero (0) then the category display will not appear in the menu. The default is 20

Table 6.4: (continued)

control	usage
initialization	Is optional and if present can have one of two values: **none** or, **normal**. If set to none the menu will always appear with the menu items set to whatever this menu file defines as the start-up state. If set to normal, the menu will be filled in with the values of the last object that had its attributes set or modified by the entry menu.

Note
In some instances, **label_display_height**, **value_display_height**, and **category_display_width** would be set to 0 so that none of these appear in the entry menu. This would be the case when setting attributes for labels, as none of these would apply and to display them would confuse the user.

The remaining lines of the attribute entry menu files refer to dynamically configured and created menu control objects. These control objects are arranged into panels, with each panel controlled by a tab. The keyword is panel followed by the tab label enclosed in quotes.

Any number of objects can be arranged within each panel. At the moment there are four types of objects defined: layout objects, attribute entry objects, control objects and display only objects.

Layout	Entry	Control	Display
frame	button_list composite_list popup_list scrolled_list spin_box spin_list text	clear_btn run_program	display_label display_attribute line

Here is a more detailed description of the objects available for use in the entry menu.

control	usage
button_list	This is a button, that when activated, pops up a list of choices. Unlike the popup_list below, there is no visual indication that the button will popup a list when activated.
clear_btn	A button, which when pressed, clears a specified list of attribute objects.
composite_list	This object contains other objects that provide input to a single attribute. Any object can be a child of a composite, but only the button_list, popup_list, scrolled_list and spin_list objects will allow input.
display_attribute	Used to display the value of a given attribute with no modification capability.
display_label	Displays a given label in the menu. Used to provide static information to users.

control	usage
frame	This is an object that encloses other object(s). It is used to "fancy up" the display. The frame can be given an optional label which is embedded in the upper left part of the frame border.
line	Displays either a horizontal or vertical line of various types.
popup_list	An object, indicated by a dash to the right of the display area, which pops up a list of choices when the mouse button is pushed down. When the mouse button is released the list pops down.
scrolled_list	An object, with a display area and an arrow displayed to the right of the area which pops up a list of choices. The list pops down when the mouse button is pushed outside of the list and selects an item if the mouse button is pushed on a list item.
spin_box	An object that displays a numeric value which is controlled by two arrows. The object must contain a value and cannot contain the idea of none.
spin_list	An object that displays a value from a defined list of values in a spin_box style object. This can contain the idea of none.
run_program	A button that when activated will run a specified program.
text	An object that allows for the free form input of text.

The list objects are for entering the value of any attribute from predefined list of values. The composite list does the same but constructs the attribute from several lists which are concatenated together. (For example the attribute value RW-SHRA could be made up of components from three lists, one of which contains RW, the second SH and the third RA. These objects are nested and related in the following manner:

```
menu_title             = title
geometry               = x y width height
label_display_height   = value
value_display_height   = value
category_display_width = value
initialization         = value
panel "panel 1 label"
  {
  entry object
    {
    keyword = value
    keyword = value
    ...
    }
  frame object
  {
  keyword = value
  keyword = value
  ...
  entry object
    {
    keyword = value
    keyword = value
    ...
    }
```

```
  }
  entry object
    {
    keyword = value
    keyword = value
    ...
    }
  }
panel "panel 2 label"
  {
  ...
  }
```

The characteristics of each of these objects are set by a list of keyword-value pairs. If part of a value string has embedded spaces this can be preserved by enclosing the string in quotes.

Example 6.1 Preserving embedded spaces

```
label     = "Cloud Amount"
position  = 241 377
size      = 500 370
```

6.7.1 Object Specification

The following points apply to most objects.

1. Most objects have a number of column settings. If not specified the program will try to set the correct number, but due to font differences you might have to set the number of columns "ncolumns" to more than you might expect.

2. In all references to colour below, the colour is entered in normal X notation. To find out more about X notation type man X at the command line. Some choices are: the X colour name (e.g. "wheat"); the rgb specification RGB: rrrr/gggg/bbbb where rrrr, gggg and bbbb are each hexadecmial digits between 0 and FFFF inclusive. (e.g. F3/DA/A9); the rgb specification RGBi: rrrr/gggg/bbbb where rrrr, gggg and bbbb are each floating point numbers between 0.0 and 1.0 inclusive. (e.g. 0.9529/0.8549/0.6627)

3. For those objects which take a list of items the following is valid:

   - The sequence of items to appear in the list must be separated by white space. If any item must have embedded spaces, then the item must be enclosed in quotes.
   - There is a special item with the name of "none" which can be inserted into the list and when selected will set the current value of the corresponding attribute to no value.
   - There is special item with the name of "range:" which can be inserted into the list. It is replaced by a range of values. The syntax for this is: ***range***:***minimum***,***maximum***,***increment***,***decimals***, where ***minimum*** and ***maximum*** are the limits to a range of numbers, ***increment*** is the amount to step between values by, and ***decimals*** is the number of decimal points to display. If decimals is not specified or is 0(zero) then the range is taken to be a range of integers. If it is specified then

the range is taken to be a range of float values. It is important that there be no spaces between the elements and the range specification must not be enclosed in quotes. There can be more than one range specification in any item list.

4. For those objects which can have their list of values limited by another object the following applies for the limit keyword:

   - There are two parameters: the limit definition and the id (attribute_id or composite_id) of the object doing the limiting.
   - The limit definition must be one of LT, LE, GE, GT.

     ```
     limit = GT cloud_base_lower
     ```

   - Some very simple logic can also be done limited to **AND** or **OR**.

     if there where two objects defining the cloud base as a range, say with composite_id's of cb1 and cb2, then we could have:

     ```
     limit = GT base:cb1 AND GT base:cb2
     ```

   - How the values in the two objects are compared depends on the limit_type key. The possible values for this are string, integer or float. This determines if the comparison is done by comparing strings, integer values or float values. The default is integer. For numerical comparisons there are several assumptions:

     (a) The values go from smaller to larger in the string.
     (b) If an entry starts with a ">" symbol it is unbounded,
     (c) and if it starts with a "<" symbol it is very small.

     See the composite_list object for information on the composite_id keyword.

The following tables give the recognized keywords and their associated values for each menu object type.

Table 6.5: Keyword list for the 'button_list' object

Keyword	Value
attribute_id	The name of the attribute to be controlled by this object.
label	If not set, then the label of the list is the attribute label. If set, the given value is used instead of the attribute label. If set to **none** then no label is displayed.
label_size	The attribute label is taken from the configuration file. There is a **long** form and **short** form label available. The default is **long**
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
foreground	The foreground colour of the object. The default is parent's.
background	The background colour of the object. The default is parent's.
border_color	The colour of the border.
border_width	The width of the border around the label. The default is 0.
margin_height	The height of the margin, in pixels, above and below the string in the label. The default is 2.
margin_width	The width of the margin, in pixels, to the right and left of the string in the label. The default is 2.

Table 6.5: (continued)

Keyword	Value
items	The sequence of items to appear in the list
nvisible	The number of items which will be visible in the scrolled list area.
ncolumns	The size of the display area specified in character columns.
none_sub	Substitute the given string for the none entry in the list. The user will see this substituted string but its action will be the same as none. The default is a blank (" ").
default	The item from the above list to be used as the default value.
limit	If the valid values in the list are limited by some other entry object this defines this relationship.
limit_type	One of **integer**, **float** or **string**. This has meaning only if a limit is specified and determines if the limit testing is numeric or by list position comparison. The default is integer.

Table 6.6: Keyword list for the 'clear_btn' object

Keyword	Value
label	The label to display to the user. The default is "«Clear"
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
foreground	The foreground colour of the object. The default is parent's.
background	The background colour of the object. The default is parent's.
border_color	The colour of the border.
border_width	The width of the border around the label. The default is 0.
margin_height	The height of the margin, in pixels, above and below the string in the label. The default is 2.
margin_width	The width of the margin, in pixels, to the right and left of the string in the label. The default is 2.
attribute_id_list	A list of attribute ids which are to be reset to their default state when this button is pressed.

Table 6.7: Keyword list for the 'composite_list' object

Keyword	Value
attribute_id	The name of the attribute to be controlled by this object.
label	If not set, then the label of the list is the attribute label. If set, the given value is used instead of the attribute label. If set to **none** then no label is displayed.
label_size	The attribute label is taken from the configuration file. There is a **long** form and **short** form label available. The default is **long**

Table 6.7: (continued)

Keyword	Value
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
border_color	The colour of the border.
border_width	The width of the border around the label. The default is 0.
margin_height	The height of the margin, in pixels, above and below the string in the label. The default is 2.
margin_width	The width of the margin, in pixels, to the right and left of the string in the label. The default is 2.
	The list of items that make up the individual parts of the composite must be specified as the item list in each child object. None of the child objects requires their attribute_id keyword to be specified and it is ignored if present. All of the children of the composite_list can have the following keywords in addition to their normal keyword set:
composite_id	If a child needs to make reference to another child, such as is required by the limit keyword, this key provides the means to give the referenced child an identifier that can be used as the reference id. If an object outside of the composite list needs to refer to an object in the composite list it can do so by using the fully qualified identifier ***attribute_id*:*composite_id***.
prefix	A character that will be inserted before that part of the composite entry controlled by the child object. If specified and the character is not a space, then the composite object will display the character before the child object. There are two special entries here that can be used for clarity: **space** which denotes the use of a space as a prefix and **dash** which denotes the use of a dash "-". All other cases are taken as literal.

Table 6.8: Keyword list for the 'display_attribute' object

Keyword	Value
attribute_id	The name of the attribute to be controlled by this object.
label	If not set, then the label of the list is the attribute label. If set, the given value is used instead of the attribute label. If set to **none** then no label is displayed.
label_size	The attribute label is taken from the configuration file. There is a **long** form and **short** form label available. The default is **long**
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
ncolumns	The size of the display area specified in character columns.

Table 6.9: Keyword list for the 'display_label' object

Keyword	Value
label	The label to display to the user.
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
foreground	The foreground colour of the object. The default is parent's.
background	The background colour of the object. The default is parent's.
border_color	The colour of the border.
border_width	The width of the border around the label. The default is 0.
margin_height	The height of the margin, in pixels, above and below the string in the label. The default is 2.
margin_width	The width of the margin, in pixels, to the right and left of the string in the label. The default is 2.

Table 6.10: Keyword list for the 'frame' object

Keyword	Value
label	The label to display on the frame which is to enclose the other objects. The default is no label.
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
size	The width and height, in pixels, of the frame object.

Table 6.11: Keyword list for the 'line' object

Keyword	Value
line_style	One of **`etched_in`** (default), **`etched_out`**, **`single`**, **`double`**, **`single_dashed`** or **`double_dashed`**.
orientation	Either **`horizontal`** (default) or **`vertical`**.
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
length	The length of the line in pixels
foreground	The foreground colour of the object. The default is parent's.
background	The background colour of the object. The default is parent's.

Table 6.12: Keyword list for the 'popup_list' object

Keyword	Value
attribute_id	The name of the attribute to be controlled by this object.
label	If not set, then the label of the list is the attribute label. If set, the given value is used instead of the attribute label. If set to **`none`** then no label is displayed.
label_size	The attribute label is taken from the configuration file. There is a **`long`** form and **`short`** form label available. The default is **`long`**
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
items	The sequence of items to appear in the list
none_sub	Substitute the given string for the none entry in the list. The user will see this substituted string but its action will be the same as none. The default is a blank (" ").
default	The item from the above list to be used as the default value.
limit	If the valid values in the list are limited by some other entry object this defines this relationship.
limit_type	One of **`integer`**, **`float`** or **`string`**. This has meaning only if a limit is specified and determines if the limit testing is numeric or by list position comparison. The default is integer.

Table 6.13: Keyword list for the 'run_program' object

Keyword	Value
label	The label to display on the button
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
size	The width and height of the button in pixels, if you want something other than the generated default size.
foreground	The foreground colour of the object. The default is parent's.
background	The background colour of the object. The default is parent's.
border_color	The colour of the border.
border_width	The width of the border around the label. The default is 0.
margin_height	The height of the margin, in pixels, above and below the string in the label. The default is 2.
margin_width	The width of the margin, in pixels, to the right and left of the string in the label. The default is 2.
program	The name of the program to run
missing	The string to pass the program if any of the parameter values is missing or not available. The default is **`none`**

Table 6.13: (continued)

Keyword	Value
parms	The run time parameters of the program. All entries are taken as literal except for special key words given in angle brackets **<*key*>**. These are substituted for the actual values when the button is activated. The keys available are: <SETUP> - The name of the setup file. <RTIME> - Run time. For depictions this is T0 <VTIME> - Valid time. <ELEMENT> - Currently active element <LEVEL> - Currently active level <LATITUDE> - Latitude of the point or label. <LONGITUDE> - Longitude of the point or label. <ATTRIB:***name***> - The value of the attribute name Latitude and longitude are only available for point fields and labels. One could have as an entry in the parms line items like latitude=<LATITUDE>

Table 6.14: Keyword list for the 'scrolled_list' object

Keyword	Value
attribute_id	The name of the attribute to be controlled by this object.
label	If not set, then the label of the list is the attribute label. If set, the given value is used instead of the attribute label. If set to **`none`** then no label is displayed.
label_size	The attribute label is taken from the configuration file. There is a **`long`** form and **`short`** form label available. The default is **`long`**
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
ncolumns	The size of the display area specified in character columns.
items	The sequence of items to appear in the list
nvisible	The number of items which will be visible in the scrolled list area.
none_sub	Substitute the given string for the none entry in the list. The user will see this substituted string but its action will be the same as none. The default is a blank (" ").
default	The item from the above list to be used as the default value.
limit	If the valid values in the list are limited by some other entry object this defines this relationship.
limit_type	One of **`integer`**, **`float`** or **`string`**. This has meaning only if a limit is specified and determines if the limit testing is numeric or by list position comparison. The default is integer.

Table 6.15: Keyword list for the 'spin_box' object

Keyword	Value
attribute_id	The name of the attribute to be controlled by this object.
label	If not set, then the label of the list is the attribute label. If set, the given value is used instead of the attribute label. If set to **none** then no label is displayed.
label_size	The attribute label is taken from the configuration file. There is a **long** form and **short** form label available. The default is **long**
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
ncolumns	The size of the display area specified in character columns.
value_max	The maximum value of the list
value_min	The minimum value of the list
default	The value which will appear in the spinbox by default
increment	The amount to increment the value by. The default is 1.
wrap	Either **True** or **False**. When the value in the spin_box reaches the maximum or minimum does it spin around to the next value. If **True**, the next value above the maximum will be the minimum and the the next value below the minimum will be the maximum. Default is **False**.

Table 6.16: Keyword list for the 'spin_list' object

Keyword	Value
attribute_id	The name of the attribute to be controlled by this object.
label	If not set, then the label of the list is the attribute label. If set, the given value is used instead of the attribute label. If set to **none** then no label is displayed.
label_size	The attribute label is taken from the configuration file. There is a **long** form and **short** form label available. The default is **long**
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
items	The sequence of items to appear in the list
ncolumns	The size of the display area specified in character columns.
none_sub	Substitute the given string for the none entry in the list. The user will see this substituted string but its action will be the same as none. The default is a blank (" ").
default	The item from the above list to be used as the default value.
limit	If the valid values in the list are limited by some other entry object this defines this relationship.
limit_type	One of **integer**, **float** or **string**. This has meaning only if a limit is specified and determines if the limit testing is numeric or by list position comparison. The default is integer.
wrap	Either **True** or **False**. When the value in the spin_list reaches the first or last item in the list does it spin around to the next value. If **True**, the next value after the last item will be the first item and the the next value before the first will be the last value. Default is **False**.

Table 6.17: Keyword list for the 'text' object

Keyword	Value
attribute_id	The name of the attribute to be controlled by this object.
label	If not set, then the label of the list is the attribute label. If set, the given value is used instead of the attribute label. If set to **`none`** then no label is displayed.
label_size	The attribute label is taken from the configuration file. There is a **`long`** form and **`short`** form label available. The default is **`long`**
position	The offset, as x y in pixels, of the entry object from the upper left corner of its parent which can be either a panel or frame.
ncolumns	The size of the text area specified in character columns. The default is 10
nrows	The vertical size of the text area in rows. The default is 1
default	A string to initialize the attribute with.

6.8 Wind Entry Menu Files

The files in the Menus directory contain definitions of attribute entry menus. The name of the file is set in the **`Elements`** block of the `Config` file. There are two types of entry files, Attribute Entry Menu files and Wind Entry Menu files. (Attribute Entry Menu files are described in the preceding section.)

The Wind Entry Menu files define the format for the information required by the wind entry menus. Each file is associated with a particular wind field or label and is specified in the `Config` file by the **`memory_file`** or **`background_memory_file`** keyword in the **`editor`** section of the **`Elements`** block.

The format of the file is rather simple for now, but is expected to increase in complexity with time as system capabilities grow.

6.8.1 File Format

The format of the entries is:

```
[header1]
  data line 1
  data line 2
  ...
[header2]
  data line 1
  ...
```

6.8.2 Header Keys

At the moment there is just one header key defined and this is **`models`** followed by the recognized model names as specified in the **`Winds`** section of the **`CrossRefs`** block in the `Config` file.

```
[models]
  Vg_Msl
  Vr_Msl
  Vc_Msl
  Vx_Msl
```

Chapter 7

Connecting to External Processes

The output of the FPA is a set of graphic, text, and/or data products. These products may be suitable "as is" for dissemination to various clients. In other cases FPA products must be sent further down the "production line" for modification or packaging. Generally FPA end products are placed somewhere in a user's FPA data directory, but the exact location can usually be specified in the "directories block" of a setup file. FPA provides hooks that allow the initiation of external processes once a product is placed in a release directory.

7.1 The Product Manager Script (fpapm)

Graphic product output programs, allied models, archiving, printing and other process which are run externally to FPA are all handled within the FPA Product Manager script (fpapm).

This script may be modified by the client site to implement procedures specific to the site. You must be aware, however, that a new release of FPA will probably contain a modified version of fpapm and that any changes made by the client will have to be made to the revised script. It is suggested that a note be made of all changes made to the script so that they can be inserted into any new release.

fpapm is located in `$FPA/bin`.

7.2 Automatically Importing Metafiles

Autoimport enables you to automatically import fields generated from an external source into the FPA. This is accomplished by putting the metafiles into a special source directory. The **`autoimport`** directory is set in the Sources block of the Config file. The default location is set in the master config file (`$FPA/config/Config.master`), but it can be reset in your local config file. When FPA detects metafiles in the import directory it asks you for permission to import the field. If, while importing a *Normal* field, a new depiction is required, FPA will ask for permission to create it.

Additional Autoimport directories can also be defined in the Sources block of the Config file, and set in the "[field.auto.import]" section of the 'interface' block of the setup file. They behave the same as the special **`autoimport`** directory.

```
Sources
  {
  ...
  autoimport
    {
  label          = <*default*> "Auto Import Directory"
  short_label    = <*default*> "Auto Import"
  source_type    = Depiction
  alias          = FPA_AutoImport
  directory_tag  = Data
  directory_path = AutoImport
  subsources     = None
  }
  ...
}
```

Appendix A

Config and Config.name File Format

A.1 Overview

The `Config` and `Config.name` files (see Configuration Files, (Chapter 6), "Configuration Files") contain information used to identify all directories, files, and fields used by the FPA.

This Appendix identifies all presently recognized keywords for information in the primary blocks of these types of configuration file. The general format of the `Config` files is:

```
revision    revision_number

include   filename1
include   filename2

block_name
  {
  member_name
    {
    keyword = value(s)
#     OR
    keyword =
      {
      keyword = value(s)
      }
    }
  }
```

In some cases, there are also ***sub_block_name*** or ***sub_member_name*** sections, but this general format is seen throughout.

The revision line must be the first uncommented line in each configuration file. (The ***revision_number*** is 8.0)

The include lines allow several configuration files to be used at the same time. The configuration file reader merges the information from the included configuration files with the information in the current configuration file to determine all parameters used by the FPA.

Note
Keyword information for any particular ***block_name*** or ***member_name*** may be included in any of the configuration files, any number of times. The information from each occurrence is merged with information already encountered to provide a full set of configuration parameters.

The general format does not allow line concatenation; for example,

```
block_name {
```

on the same line would not be recognized.

Note
In order to make this appendix easier to read the following conventions were adopted:

- when a word is being used as a place holder it is formatted to indicate it should be ***replaced*** by an appropriate value.
- if a keyword only accepts values from a predefined list, the list is separated with '|' to indicate "or" or '&|' to indicate "and/or".
- Sub-blocks are indented
- The values for some keywords can only be specified once. If such a keyword has additional specifications, only the first one is accepted. The rest are ignored. (Those keywords are preceded by a '!' in the following sections.)

Note
The format of the actual configuration files is important as the parsing program can be rather unforgiving.

- Line continuations (where necessary) are indicated by ending the line with '\'.
- Comment lines begin with '#', but the format also allows for appended comments on most lines.
- The values given for keywords can usually be specified as often as desired, with each new value replacing the previous specification. (Note that "language" specific keywords are a special case, described in more detail below.) In those cases where the keyword defines a list of values, each new value is added to the present list; there is usually a "reset_" keyword to allow the user to remove all previous specifications.
- The values for some keywords refer to members of other blocks. These members must therefore be defined in one of the `Config` files used.
- There are several "magic" values for keywords (examples are Default or None), which have a special meaning to the programs. These magic values should not be used as members in any block in the `Config` files.
- All '{' or '}' characters must be located on their own lines.
- The '=' must be separated from the rest of the line by white space.

Some changes to the `Config` files are not allowed, or not presently supported. Attempts to make such changes will result in a warning message in the log file, but should not affect the running of the FPA. Errors in a line of a `Config` file will cause an error for that member of the block, and will print out an error message in the log file. The affected member will no longer be recognized by the FPA, and will no longer appear in any list!

Warning
Users should NEVER attempt to edit a `Config` file when the FPA is running, as this may have a catastrophic effect on the FPA and its data files.

A.2 Support for Other Languages

There are two independent portions of the FPA where the language can be controlled locally. One is the set of menu selections (upper left corner of the FPA windows) and informative messages displayed by the FPA (displayed in the upper corner of the depiction window, just below these buttons.) These items make up the 'interface' proper, and the only aspect which is accessible by users is the text displayed on the buttons or messages. The `FPA_LANG` setting in the UNIX environment controls the language in the FPA interface through a series of language files in subdirectories of `$FPA/app-defaults`.

The other portion of the FPA that is language dependent is the set of buttons on the right hand side of the FPA window. These buttons can be altered and configured within the configuration files, as discussed below.

A.2.1 Other Languages in the Configurable Menus

A language specifier is allowed in all lines with the format ***keyword*** = ***label*** in the FPA configuration files (for example, label = ***label*** or short_label = ***label***). The format of language dependent labels is given by:

```
keyword = <*default*> "default label" \
    <*second_language*> "second language label" \
    <*third_language*>  "third language label"
#     OR
keyword = <*default*>          "default label"
keyword = <*second_language*> "second language label"
keyword = <*third_language*>  "third language label"
```

For example, labels for mean sea level pressure in English and Canadian French (indicated by c-french in UNIX) could be entered as:

```
label = <*default*> "MSL Pressure" <*c-french*> "Pression MSL"
```

In this example, if the `FPA_LANG` environment variable is set to c-french, "Pression MSL" would be displayed in the FPA interface.

As noted above, keywords can appear several times in different config files, but the final entry in the configuration files normally takes precedence. However, keywords that allow a language specifier are special cases, and only lines containing the language set by the `FPA_LANG` environment variable are recognized. This allows keywords for different languages to be contained in separate configuration files. The default value for each label is used if no "language" specific value is encountered!

A.3 Common Keywords

The following table describes keywords that are common to several configuration file blocks, or which contain lists of preset choices for keywords. A description of each of the preset choices is also given. Occurrences of these keywords in the individual configuration file blocks described below will refer back to this Table.

Table A.1: Common Keywords

Keyword	Explanation
label	This is the label the user sees.
short_label	This is the label the user sees when there is not much space on the button. It should be less than 12 characters
description	This is an optional descriptive label
file_id	This is the identifier used to construct (old style) FPA file names. (element identifiers are limited to 2 characters!) For example, in the `Config.master` file, pressure has a file_id of pn. Thus metafiles containing pressure data are named with the prefix pn (e.g. a depiction metafile named pnmsl_1998:182:00 for mean sea level pressure).
file_ident	This is the identifier used to construct (new style) FPA file names (limited to 32 characters.) May not contain spaces or '~')
field_group	This parameter will group the FPA fields in the interface. The group name must be from the 'Fields' sub block of the 'Groups' block.
element_group	This parameter will group the FPA elements in the interface. The group name must be from the 'Elements' sub block of the 'Groups' block
field_type	This parameter identifies the type of meteorological field. One of: **Continuous**: for contoured b-spline fields (grid point data). **Vector**: for u, v component b-spline fields **Discrete**: for area fields **Wind**: for fields containing areas of adjusted wind **Line**: for line fields **LChain**: for link chain fields **Scattered**: for data at random points **Special**: for all other fields

Table A.1: (continued)

Keyword	Explanation
level_type	This parameter matches acceptable elements and levels to create fields. One of: **Msl**: for mean sea level fields **Surface**: for surface of the Earth fields **Level**: for level type fields **Layer**: for layer type fields **Geography**: for geographical fields **Annotation**: for menus or legends **Any**: for special elements that may be valid at any level
source_type or **source_types**	This parameter sets the type of files for a data source. One of: **Depiction**: for data sources where all fields are unique **Guidance**: for data sources where fields may be reproduced at a different run time for the data source **Allied**: for data sources from Allied Models (these are similar to Guidance type data sources) **Maps**: for geographical data sources **Direct**: (not yet used) **Any**: for any of the above types
value_function	This parameter sets a value calculation function used to determine a given field. Note that each value calculation function refers to an FPA routine to calculate a value based on given input fields. One of: **FPA_Value_Func**: extract values from a single field **FPA_Vector_Magnitude_Func**: extract a magnitude from a vector field **FPA_Vector_Direction_Func**: extract a direction from a vector field **FPA_Daily_Max_Value/Time_Func**: extract maximum values or times of maximum values from a series of fields **FPA_Daily_Min_Value/Time_Func**: extract minimum values or times of minimum values from a series of fields **FPA_Get_Daily_Value_Func**: extracts values from a daily field that matches a given time. **FPA_Actual_Wind_Speed/Direction_Func**: extract wind speed or wind direction from a field containing areas of adjusted wind Note that user defined value functions can also be used.

Table A.1: (continued)

Keyword	Explanation
wind_function	This parameter sets a wind calculation function used to determine a given type of wind. Note that each wind calculation function refers to an FPA routine to calculate wind speed and direction based on given input fields. One of: **FPA_Geostrophic_Wind_Func**: extract wind based on the geostrophic wind equation **FPA_Gradient_Wind_Func**: extract wind based on the gradient wind equation **FPA_Cyclostrophic_Wind_Func**: extract wind based on the cyclostrophic wind equation **FPA_Isallobaric_Wind_Func**: extract wind based on the isallobaric wind equation **FPA_Thermal_Wind_Func**: extract wind based on the thermal wind equation **FPA_UVcomponent_Wind_Func**: extract wind based on the u and v components of wind **FPA_DirectionSpeed_Wind_Func**: extract wind based on the direction and speed components of wind **FPA_Adjusted_Wind_Func**: extract wind from the wind adjustment field where **field_type = Wind** **FPA_Absolute_Wind_Func**: set absolute wind speed and direction to zero. Note that user defined wind functions can also be used.
value_sample	This parameter determines the type of value to sample from a field. Note that each type of value refers to an FPA routine to determine parameters from a given input field. One of: **FPA_Sample_Value**: sample the value of Continuous or Vector type field **FPA_Sample_Gradient**: sample the gradient of a Continuous type field **FPA_Sample_Curvature**: sample the curvature of a Continuous type field **FPA_Sample_Magnitude**: sample the Magnitude of a Vector type field **FPA_Sample_Direction**: sample the direction of a Vector type field
value_sample_types	This parameter sets value sampling types for a given field. Each ***sample_type_name*** is from the 'Values' sub block of the 'Samples' block.
wind_sample_type or **wind_sample_types**	This parameter sets wind sampling types for a given field. Each ***sample_type_name*** is from the 'Winds' sub block of the 'Samples' block

A.4 Elements block

An element is a meteorological parameter that exists at one or more levels. The 'Elements' block is used to identify information specific to an element in the FPA.

Note that ***group_name*** must be a member of the 'Elements' sub block of the 'Groups' block, or the 'Fields' sub block of the 'Groups' block.

Note that ***crossref_name*** must be a member of the 'Winds' sub block of the 'CrossRefs' block, or the 'Values' sub block of the 'CrossRefs' block.

Note that ***sample_type_name*** must be a member of the 'Values' sub block of the 'Samples' block, or the 'Winds' sub block of the 'Samples' block.

Note that ***unit_name*** must be a member of the 'Units' block.

A.4.1 Template of an 'Elements' Block

Elements
{

element_name
{

label = ***label***
This is the label that the user sees. The default is ***element_name***. This parameter is "language" sensitive!

short_label = ***label***
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***element_name***. This parameter is "language" sensitive!

description = ***label***
This is an optional descriptive label. This parameter is "language" sensitive!

! field_type =
Must be one of: **Continuous**, **Vector**, **Discrete**, **Wind**, **Line**, **LChain**, **Scattered**, **Special**
This parameter identifies the type of meteorological field. The choices are described in Table A.1.
The choices available for the 'editor', 'labelling', and 'sample' keywords (below) are determined by the field_type. This is indicated in the comments associated with each keywords. Note that no 'editor', 'labelling', or 'sample' keywords are available for Special elements.
Note that this parameter cannot be re-specified!

! level_type =
Must be one of: **Msl**, **Surface**, **Level**, **Layer**, **Geography**, **Annotation**, **Any**
This parameter matches the element to an acceptable level to create a field. The choices are described in Table A.1.
Note that this parameter cannot be re-specified!

Note
The following **display_format** keyword is applicable to **Discrete** fields.

display_format = Simple | Complex
This parameter should be set to **Complex** only if the **Discrete** field will have both dividing lines AND holes. A **Discrete** field with both dividing lines and holes requires more complicated calculations to ensure correct masking and display of features.

element_group = *group_name*
This parameter groups FPA elements in the interface. The ***group_name*** is from the 'Elements' sub block of the 'Groups' block. The default group is 'Miscellaneous'.

field_group = *group_name*
This parameter groups FPA fields in the interface. The ***group_name*** is from the 'Fields' sub block of the 'Groups' block. This parameter is the default group if the field or the field_group for the field are not specified.

alias = *alias_name(s)*
These can be shorter names for use, say, in equations. (An example is the label "Pressure" which has a short_label "Pres". Aliases include "p" and "pres". These aliases can be used in equations which involve pressure.) Alias names cannot be reused by another element. Only the first one encountered gets used.

! file_id = None | *identifier*
This is the identifier used to construct old style FPA file names (see Table A.1). It cannot be re-specified unless changed from **None**!

! file_ident = None | *identifier*
This is the identifier used to construct new style FPA file names (see Table A.1). It cannot be re-specified unless changed from **None**!

precision = *value unit_name*
This is the output precision of the element for metafiles (usually 0.01) and the normal units for the element. This value is required for Continuous fields.

time_dependence = Normal
OR

time_dependence =
{
This parameter identifies temporal information for the field.

time_type =
Sets the time coverage of the field. Must be one of:
- **Normal**: for fields valid at one time (such as hourly temperature).
- **Daily**: for fields valid over a time period (such as the daily maximum temperature).
- **Static**: for fields that remain unchanged in time (such as ice cover).

daily_range = *normal_time begin_time end_time unit_name*
This parameter sets times for Daily fields; the normal time of day for the field, the begin and end times for display of the field, and the units that the times are in.

} #(End of time_dependence Block)

wind_class =
Identifies the type of wind calculation appropriate for the meteorological field. Must be one of:
- **Pressure**: for pressure or pressure change fields.
- **Height**: for geopotential height fields.
- **Thickness**: for geopotential thickness fields.
- **Adjustment**: for wind adjustment fields.
- **None**: for fields with no appropriate type of wind calculation.

attributes_reset = True | False
This parameter resets the attributes list given below.

attributes = Default
OR

attributes =
This parameter identifies allowed attributes for the field. Attributes are parameters that can be attached to objects in Discrete, Wind, Line, or LChain type fields, or attached to labels in any field.
Some default attributes are automatically declared by the FPA, and therefore do not need to be declared in any configuration file. However, the default value of those attributes with the prefix "FPA_" may be reset in this section, or in the labelling block described below. Also, the values for default attributes with the prefix "EVAL_" are automatically calculated from the FPA fields. (Note that they are most often used for field labels, and are set with a default_value of Yes in the attribute_defaults section of the labelling block described below.)
Default attributes for all fields
- **FPA_user_label**: for a user defined label
- **FPA_auto_label**: for an automatic label
- **FPA_category**: for a category of feature
- **FPA_label_type**: for a type of label
Default attributes for **Continuous** and **Vector** fields
- **EVAL_contour**: for the value on the closest field contour
- **EVAL_spval**: for the value at a set location on the field
- **EVAL_wind**: for a wind evaluated at a set location
Default attributes for **Wind** fields
- **FPA_wind_model**: for the default wind calculation model
- **FPA_wind_direction**: for the default wind direction adjustment
- **FPA_wind_speed**: for the default wind speed adjustment
- **FPA_wind_gust**: for the default wind gust adjustment
- **EVAL_wind**: for a wind evaluated at a set location
Default attributes for **Line** fields
- **FPA_line_type**: for the type of line
- **FPA_proximity**: proximity of sample to line (kilometres)
- **FPA_line_direction**: direction of line at sample point (degrees true)
- **FPA_line_length**: length of the line in (kilometres)
Default attributes for **LChain** fields
- **FPA_lchain_reference**: T0 associated with link chain.
- **FPA_lchain_start_time**: time wrt **FPA_lchain_reference** of first node in

link chain.
- **FPA_lchain_end_time**: time wrt **FPA_lchain_reference** of last node in link chain.
Default attributes for **LChain nodes**
- **FPA_lnode_type**: type of node one of ...
- **FPA_lnode_time**: time wrt **FPA_lchain_reference** of node.
- **FPA_lnode_direction**: direction of movement calculated from current link node to the next node in the link chain.
- **FPA_lnode_speed**: speed calculated from current link node to the next link node in the link chain.
- **FPA_lnode_vector**: direction and speed calculated from current link node to the next link node in the link chain.
Default attributes for **Scattered** fields
- **FPA_scattered_type**: for the type of line
- **FPA_proximity**: proximity of sample to scattered point (kilometres)

Note that, unless **attributes_reset = True** is used, attributes defined below are ADDED to those already defined.
{
attribute_name [*default_value*]
Note that if only the default_value is required, it can be set on this line!.
{
attribute_label = *label*
This is the label that the user sees. The default is attribute_name. This parameter is "language" sensitive!
attribute_short_label = *label*
The attribute_short_label is used when there is not enough room on the FPA screen to use the longer attribute_label. The default is attribute_name. This parameter is "language" sensitive!
attribute_background_default = *default_value*
This parameter identifies the default value for the attribute. Note that the ***default_value*** can also be set in an entry menu (described in the entry_file or background_entry_file keywords in the editor block below). The ***default_value*** in these files takes precedence over any value set here!
} #(End of attribute_name Block)
} #(End of attributes Block)

Note
The following **line_types** keywords are applicable to **Line** fields.

line_types_reset = True | False
This parameter resets the line_types list given below.

line_types = None
OR

line_types =
This parameter identifies allowed line types for the field. The line_types keyword is only applicable to **field_type = Line**.
Note that, unless **line_types_reset = True** is used, subelements are ADDED to those already defined.

{
line_type_name
{
type_label = ***label***
This is the label that the user sees. The default is ***label_type_name***. This parameter is "language" sensitive!
type_short_label = ***label***
The type_short_label is used when there is not enough room on the FPA screen to use the longer type_label. The default is ***label_type_name***. This parameter is "language" sensitive!
pattern = ***filename***
This is an FPA metafile describing the line pattern. It will appear in the drawing selection list for the **field_type = Line** field editor.
} #(End of line_type_name Block)
} #(End of line_types Block)

Note
The following **scattered_types** keywords are applicable to **Scattered** fields.

scattered_types_reset = True | False
This parameter resets the scattered_types list given below.

scattered_types = None
OR

scattered_types =
This parameter identifies allowed scattered types for the field. The scattered_types keyword is only applicable to **field_type = Scattered**.
Note that, unless **scattered_types_reset = True** is used, scattered types are ADDED to those already defined.

{
scattered_type_name
{

type_label = *label*
This is the label that the user sees. The default is scattered_type_name. This parameter is "language" sensitive!

type_short_label = *label*
The type_short_label is used when there is not enough room on the FPA screen to use the longer type_label. The default is ***label_type_name***.This parameter is "language" sensitive!

type_class = *class_name*
This parameter sets a label category.

type_entry_file = *entry_file_name*
Identifies an entry menu file to be accessed when a label is added or modified. Note that self-documented example menus can be found in the directory `$FPA/config/Menus`.

type_modify_file = *entry_modify_name*
Identifies an entry menu file to be accessed when a label is modified. The default is type_entry_file above. Note that self-documented example menus can be found in the directory `$FPA/config/Menus`.

type_attach =
This parameter controls where the label appears on the field. Must be one of:
- **attach_auto**: for the closest point
- **attach_point**: for the closest point

attribute_defaults_reset = True | False
This parameter resets the attribute_defaults list given below.

attribute_defaults =
This parameter sets default attribute values for this type of label.
Note that, unless **attribute_defaults_reset = True** is used, default attribute values are ADDED to those already defined. Note that default attributes which can be set here (including those with the prefix "EVAL_") are described in the attributes block above.
{
attribute_name default_value
}

type_rules_reset = True | False
This parameter resets the type_rules list given below.

type_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for this label are set.
Note that, unless **type_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

python_type_rules_reset = True | False
This parameter resets the python_type_rules list given below.

python_type_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA python entry rule functions that will be invoked when attributes for this label are set.

Note that, unless **python_type_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

} #(End of scattered_type_name Block)

} #(End of scattered_types Block)

editor = None

OR

editor =

This parameter identifies information for editing the field.

{

Note

The following keywords are applicable to all field editors.

---

merge_fields_reset = True | False

This parameter resets the merge_fields list given below

merge_fields = *element level*

Identifies fields that can be used in the *Merge* editor tool, so that features can be copied from another FPA field. Note that the fields must have units that can be converted to those of the FPA field.

Note

The following keywords are applicable to **Continuous** field editors.

---

hilo = True | False

Determines whether maxima and minima of the field will be shown.

poke = *value unit_name*

Identifies the magnitude and units used on the sidebar for field edits. A poke value is required if you want this element to be editable.

Note

The following keywords are applicable to **Vector** field editors.

---

hilo = True | False

Determines whether maxima and minima of the field will be shown.

magnitude_poke = *value unit_name*

Identifies the magnitude and units used on the sidebar for field edits.

direction_poke = *value unit_name*

Identifies the magnitude and units used on the sidebar for field edits.

Note
The following keywords are applicable to **Wind** field editors.

display_order = True | False
Show numbered boxes to display the stacking order of areas drawn.

entry_file = *entry_file_name*
Identifies an entry menu file to be accessed when the "Set" button on the sidebar is clicked. Note that self-documented example menus can be found in the directory `$FPA/config/Menus`.

modify_file = *modify_file_name*
Identifies an entry menu file to be accessed when the "Set" button for a modified area is clicked. The default is entry_file_name above.

memory_file = *memory_file_name*
Identifies a predefined list of choices for applying to drawn areas. Note that self-documented example memory lists can be found in the directory `$FPA/config/Memory`.

background_entry_file = *background_entry_file_name*
Identifies an entry menu file to be accessed when the background "Set" button on the sidebar is clicked. Note that self-documented example background menus can be found in the directory `$FPA/config/Menus`. (The format is identical to entry_file.)

background_memory_file = *background_memory_file_name*
Identifies a predefined list of choices for applying as the background value for the given field. Note that self-documented example background memory lists can be found in the directory `$FPA/config/Memory`. (The format is identical to memory_file.)

entry_rules_reset = True | False
This parameter resets the entry_rules list given below.

entry_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for any entry_file, modify_file or memory_file are set.
Note that, unless **entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

python_entry_rules_reset = True | False
This parameter resets the python_entry_rules list given below.

python_entry_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions, written in python, that will be invoked when attributes for any entry_file, modify_file or memory_file are set.
Note that, unless **python_entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

Note
The following keywords are applicable to **Discrete** field editors.

display_order = True | False
Show numbered boxes to display the stacking order of areas drawn.

entry_file = *entry_file_name*
Identifies an entry menu file to be accessed when the "Set" button on the sidebar is clicked. Note that self-documented example menus can be found in the directory `$FPA/config/Menus`.

modify_file = *modify_file_name*
Identifies an entry menu file to be accessed when the "Set" button for a modified area is clicked. The default is entry_file_name above.

memory_file = *memory_file_name*
Identifies a predefined list of choices for applying to drawn areas. Note that self-documented example memory lists can be found in the directory `$FPA/config/Memory`.

background_entry_file = *background_entry_file_name*
Identifies an entry menu file to be accessed when the background "Set" button on the sidebar is clicked. Note that self-documented example background menus can be found in the directory `$FPA/config/Menus`. (The format is identical to entry_file.)

background_memory_file = *background_memory_file_name*
Identifies a predefined list of choices for applying as the background value for the given field. Note that self-documented example background memory lists can be found in the directory `$FPA/config/Memory`. (The format is identical to memory_file.)

entry_rules_reset = True | False
This parameter resets the entry_rules list given below.

entry_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for any entry_file, modify_file or memory_file are set.
Note that, unless **entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

python_entry_rules_reset = True | False
This parameter resets the python_entry_rules list given below.

python_entry_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions, written in python, that will be invoked when attributes for any entry_file, modify_file or memory_file are set.
Note that, unless **python_entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

overlaying = True | False
Determines whether the overlapping areas can both be sampled.

Note
The following keywords are applicable to **Line** field editors.

entry_file = *entry_file_name*
Identifies an entry menu file to be accessed when a line type is drawn

modify_file = *modify_file_name*
Identifies an entry menu file to be accessed when a line type is modified. The default is ***entry_file_name***

entry_rules_reset = True | False
This parameter resets the entry_rules list given below.

entry_rules = *entry_rules_name(s)*
Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for any entry_file or modify_file are set.
Note that, unless **entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

python_entry_rules_reset = True | False
This parameter resets the python_entry_rules list given below.

python_entry_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions, written in python, that will be invoked when attributes for any entry_file or modify_file are set.
Note that, unless **python_entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

Note
The following keywords are applicable to **Scattered** field editors.

entry_file = *entry_file_name*
Identifies an entry menu file to be accessed when a scattered type is drawn

modify_file = *modify_file_name*
Identifies an entry menu file to be accessed when a scattered type is modified. The default is ***entry_file_name***

entry_rules_reset = True | False
This parameter resets the entry_rules list given below.

entry_rules = *entry_rules_name(s)*
Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for any entry_file or modify_file are set.
Note that, unless **entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

python_entry_rules_reset = True | False
This parameter resets the python_entry_rules list given below.

python_entry_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions, written in python, that will be

invoked when attributes for any entry_file or modify_file are set.
Note that, unless **python_entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

Note
The following keywords are applicable to **LChain** field editors.

---

entry_file = *entry_file_name*
Identifies an entry menu file to be accessed when a lchain type is drawn

modify_file = *modify_file_name*
Identifies an entry menu file to be accessed when a lchain type is modified. The default is ***entry_file_name***

entry_rules_reset = True | False
This parameter resets the entry_rules list given below.

entry_rules = *entry_rules_name(s)*
Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for any entry_file or modify_file are set Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for any entry_file or modify_file are set.
Note that, unless **entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

python_entry_rules_reset = True | False
This parameter resets the python_entry_rules list given below.

python_entry_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions, written in python, that will be invoked when attributes for any entry_file or modify_file are set.
Note that, unless **python_entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

interpolation_delta = *HH:mm*
Sets the default time between nodes when adding a link chain.

node_entry_file = *entry_file_name*
Identifies an entry menu file to be accessed when a link chain node is placed.

node_modify_file = *modify_file_name*
Identifies an entry menu file to be accessed when a link chain node is modified. The default is ***entry_file_name***

node_entry_rules_reset = True | False
This parameter resets the node_entry_rules list given below.

node_entry_rules = *entry_rules_name(s)*
Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for any node_entry_file or node_modify_file are set Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for any node entry_file or modify_file are set.

Note that, unless **node_entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

python_node_entry_rules_reset = True | False
This parameter resets the python_node_entry_rules list given below.

python_node_entry_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions, written in python, that will be invoked when attributes for any node_entry_file or node_modify_file are set.
Note that, unless **python_node_entry_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

link_fields_reset = True | False
This parameter resets the link_fields list given below

link_fields = *element level*
Identifies fields that can be used in the *Merge* editor tool, so that link chains used to timelink another FPA field can be copied to this field.

} #(End of editor Block)

labelling_reset = True | False
This parameter resets the labelling list to default values.

labelling = Default
OR

labelling =
This parameter identifies information for labelling the field. The labelling keyword is NOT applicable to **field_type = Scattered**.
{

label_types_reset = True | False
This parameter resets the label_types list given below to default values.

label_types =
This parameter identifies allowed types of labels for the field.
Note that, unless **label_types_reset = True** is used, label types are ADDED to those already defined.
{

label_type_name
{

type_label = *label*
This is the label that the user sees. The default is ***label_type_name***. This parameter is "language" sensitive!

type_short_label = *label*
The type_short_label is used when there is not enough room on the FPA screen to use the longer type_label. The default is ***label_type_name***. This parameter is "language" sensitive!

type_class = *class_name*
This parameter sets a label category.

type_entry_file = *entry_file_name*
Identifies an entry menu file to be accessed when a label is added. Note that self-documented example menus can be found in the directory `$FPA/config/Menus`.

type_modify_file = *entry_modify_name*
Identifies an entry menu file to be accessed when a label is modified. The default is ***type_entry_file*** above. Note that self-documented example menus can be found in the directory `$FPA/config/Menus`.

type_attach =
This parameter controls where the label appears on the field.
Acceptable parameters for **field_type = Continuous** or **Vector** fields are:
- **no_attach**: for the current location (when chosen)
- **attach_auto**: for the closest minimum, maximum, or contour
- **attach_min**: for the closest minimum on the field
- **attach_max**: for the closest maximum on the field
- **attach_col**: for the closest saddle point on the field
- **attach_contour**: for the closest contour on the field

Acceptable parameters for **field_type = Discrete** or **Wind** fields are:
- **attach_auto**: for the current location (when chosen)

Acceptable parameters for **:field_type = Line** fields are:
- **attach_auto**: for the closest line
- **attach_line**: for the closest line

attribute_defaults_reset = True | False
This parameter resets the attribute_defaults list given below.

attribute_defaults =
This parameter sets default attribute values for this type of label.
Note that, unless **attribute_defaults_reset = True** is used, default attribute values are ADDED to those already defined.
Note that default attributes which can be set here (including those with the prefix "EVAL_") are described in the attributes block above.
Note that if a label is to have extra attributes that are not part of the object to which the label is attached, then those attributes must be listed here. Otherwise, any modifications of the object will replace the value of the extra attribute with a blank!
{
attribute_name default_value
}

type_rules_reset = True | False
This parameter resets the type_rules list given below.

type_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA entry rule functions that will be invoked when attributes for this label are set.
Note that, unless **type_rules_reset = True** is used, entry rule functions are ADDED to those already defined.
Note that user defined entry rule functions can also be used.

python_type_rules_reset = True | False
This parameter resets the python_type_rules list given below.

python_type_rules = *entry_rule_name(s)*
Identifies a list of name(s) for FPA python entry rule functions that will be invoked when attributes for this label are set.
Note that, unless **python_type_rules_reset = True** is used, entry rule functions are ADDED to those already defined. Note that user defined entry rule functions can also be used.

} #(End of label_type_name Block)

} #(End of label_types Block)

} #(End of labelling Block)

sampling_reset = True | False
The parameter resets the sample list below to its default values.

sample = Default
OR

sample =
This parameter identifies information for sampling the field.

{

Note
The following keywords are applicable to **Continuous** or **Vector** field editors.

---

value_sample_types = *sample_type_name(s)*
This parameter sets the types of values to sample from a given field.
Each ***sample_type_name*** is from the Values sub block of the 'Samples' block.

wind_sample_types = *sample_type_name(s)*
This parameter sets the types of winds to sample from a given field.
Each ***sample_type_name*** is from the 'Winds' sub block of the 'Samples' block.

Note
The following keywords are applicable to **Discrete**, **Line**, **LChain** or **Scattered** field editors.

---

attribute_sample_names = *attribute_name(s)*
This parameter sets the names of attributes to sample from a given field.
Each ***attribute_name*** is from the attributes sub block above..

Note
The following keywords are applicable to **Wind** field editors.

---

wind_crossrefs = *crossref_name(s)*
This parameter sets the types of winds to sample based on other cross-referenced fields.
Each ***crossref_name*** is from the 'Winds' sub block of the 'CrossRefs' block.

wind_sample_type = *sample_type_name*
This parameter sets the adjusted wind to sample from a given wind field.
The ***sample_type_name*** is from the 'Winds' sub block of the 'Samples' block, and must use wind_function = FPA_Adjusted_Wind_Func.

} #(End of sample Block)

equation = None
OR
equation =
This parameter is used to identify an equation for calculating the given meteorological field from other meteorological fields with a mathematical equation. See Equation Evaluator, (Appendix B) for more information about equations.

{

force_calculation = True | False
This parameter should be set to True for testing new equations. It forces a recalculation of the equation from the input fields.

equation_string = *equation*
This parameter gives the equation for the meteorological field. The format of equations is explained in Equation Evaluator, (Appendix B).

equation_units = *unit_name*
This parameter gives the units for the equation.

} #(End of equation Block)

value_calculation = None
OR
value_calculation =
This parameter is used to identify a method of determining the given meteorological field from other meteorological fields with a cross-reference.

{

force_calculation = True | False
This parameter should be set to True for testing new equations. It forces a recalculation of the equation from the input fields.

value_crossref = *crossref_name*
This parameter identifies the cross-reference for determining the meteorological field. The ***crossref_name*** is from the 'Values' sub block of the 'CrossRefs' block.

source_types =
This parameter sets the types of data sources for determining the meteorological field. More than one can be specified. The choices are described in Table A.1. May be one or more of: **Depiction**, **Guidance**, **Allied**, **Maps**, **Direct**, **NotUsed**.

} #(End of value_calculation Block)

components = None
OR
components =
This parameter is used to identify meteorological fields that have u and v (x and y) components, or vector (direction and magnitude) components. Both components must be identified.
{
x_component = *element_name*
y_component = *element_name*
These parameters identify the u and v (x and y) components of a meteorological field.
OR
direction_component = *element_name*
magnitude_component = *element_name*
These parameters identify the vector (direction and magnitude) components of a meteorological field.
} #(End of components Block)

linking_reset = True | False
The parameter resets the link list below to its default values.
linking = Default
OR
linking =
You can share time links between fields by including them here.
{
link_fields_reset = True | False
The parameter resets the **link_fields** list below to empty.
link_fields = *element_name level_name*
This parameter may be specified more than once to build a list of fields (element/level pairs) from which link chains can be imported.
} #(End of **linking** Block)

} #(End of ***element_name*** Block)

} #(End of 'Elements' Block)

A.5 Fields block

A field is an element at a specific level. The 'Fields' block is used to identify information specific to a field in the FPA. Note that a field accessed in the FPA need not be declared as a field in the 'Fields' block. Only the element and level need to be declared; the FPA will automatically create a field for any given element and level using the default values. An entry in the 'Fields' block is only required if the user wishes to override the information for a specific element and level.

Note that ***element_name*** must be a member of the 'Elements' block, and that ***level_name*** must be a member of the 'Levels' block.

Note that ***group_name*** must be a member of the 'Fields' sub block of the 'Groups' block.

A.5.1 Template of a 'Fields' Block

Fields
{

element_name level_name
{

label = ***label***
This is the label that the user sees in the FPA menus. The default is the ***level_name*** label followed by the ***element_name*** label. This parameter is "language" sensitive!

short_label = ***label***
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is the ***level_name*** short_label followed by the ***element_name*** short_label. This parameter is "language" sensitive!

field_group = ***group_name***
This parameter will group the FPA fields in the interface. The ***group_name*** is from the 'Fields' sub block of the Groups block. The default group is 'Miscellaneous'.

element_info = Default
OR
element_info =
{
attributes_reset = True | False

attributes =
{
(see attributes keyword in 'Elements' block for acceptable keywords)
}

Note
The following **line_types** keywords are applicable to **Line** fields.

---

line_types_reset = True | False

line_types =
{
(see line_types keyword in 'Elements' block for acceptable keywords)
}

Note
The following **scattered_types** keywords are applicable to **Scattered** fields.

```
        scattered_types_reset = True | False

        scattered_types =
            {
            (see scattered_types keyword in 'Elements' block for acceptable keywords)
            }

        editor =
            {
            (see editor keyword in 'Elements' block for acceptable keywords)
            }

        labelling_reset = True | False

        labelling =
            {
            (see labelling keyword in 'Elements' block for acceptable keywords)
            }

        sampling_reset = True | False

        sample =
            {
            (see sample keyword in 'Elements' block for acceptable keywords)
            }

        equation =
            {
            (see equation keyword in 'Elements' block for acceptable keywords)
            }
        value_calculation =
            {
            (see value_calculation keyword in 'Elements' block for acceptable keywords)
            }
        linking =
            {
            (see linking keyword in 'Elements' block for acceptable keywords)
            }
        } #(End of element_info Block)

    level_info = Default

    } #(End of element_name level_name Block)

} #(End of 'Fields' Block)
```

A.6 Levels block

The 'Levels' block is used to identify information specific to a level in the FPA where level is a level in the atmosphere at which meteorological parameters are valid.

Note that ***group_name*** must be a member of the 'Fields' sub block of the Groups block.

A.6.1 Template of a 'Levels' Block

Levels
{

level_name
{

label = ***label***
This is the label that the user sees. The default is ***level_name***. This parameter is "language" sensitive!

short_label = ***label***
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***level_name***. This parameter is "language" sensitive!

description = ***label***
This is an optional descriptive label. This parameter is "language" sensitive!

! level_type =
This parameter matches the level to an acceptable element to create a field. The choices are described in Table A.1. Must be one of: **Msl**, **Surface**, **Level**, **Layer**, **Geography**, **Annotation**, **Any**. Note that this parameter cannot be re-specified!

field_group = ***group_name***
This parameter will group the FPA fields in the interface. The ***group_name*** is from the 'Fields' sub block of the Groups block. This parameter is the default group if the field or the field_group for the field or the field_group for the element of the field are not specified.

alias = ***alias_name(s)***
These can be shorter names for use in equations, for example. Alias names cannot be reused by another level. Only the first one encountered gets used.

! file_id = None | ***identifier***
This is the identifier used to construct old style FPA file names (see Table A.1). It cannot be re-specified unless changed from **None**!

! file_ident = None | ***identifier***
This is the identifier used to construct new style FPA file names (see Table A.1). It cannot be re-specified unless changed from **None**!

! level_levels = ***category***
OR

! level_levels = ***category value***
OR

! **level_levels =** ***category upper lower***

This parameter identifies information used to set the vertical location in the atmosphere of the given ***level_name***, where:
- ***category*** is one of: **Msl**, **Surface**, **Pressure**, **Height**, **Sigma**, **Theta**, **Geography**, **Annotation**.
- ***value*** is a single numeric value for a level
- ***upper lower*** are two numeric values for a layer
Note that this parameter cannot be re-specified!
The acceptable value for **level_type = Msl** is:
- **level_levels = Msl** (for mean sea level)
The acceptable value for **level_type = Surface** is:
- **level_levels = Surface** (for the surface of the Earth)
The acceptable values for **level_type = Level** are:
- **level_levels = Pressure** ***value*** (for a constant pressure surface)
- **level_levels = Height** ***value*** (for a constant height surface)
- **level_levels = Sigma** ***value*** (for a sigma surface)
- **level_levels = Theta** ***value*** (for an isentropic surface)
The acceptable values for **level_type = Layer** are:
- **level_levels = Pressure** ***upper lower*** (for a pressure layer)
- **level_levels = Height** ***upper lower*** (for a height layer)
- **level_levels = Sigma** ***value*** (for a sigma layer)
- **level_levels = Theta** ***value*** (for an isentropic layer)
The acceptable value for **level_type = Geography** is:
- **level_levels = Geography** (for geographical information)
The acceptable value for **level_type = Annotation** is:
- **level_levels = Annotation** (for menus or legends)

} #(End of ***level_name*** Block)

} #(End of 'Levels' Block)

A.7 Groups block

The Groups block is used to identify grouping of fields and elements in the FPA interface. The 'Fields' ***group_name*** "Not_Displayed" identifies fields that will not appear in the interface, the 'Elements' ***group_name*** "Not_Displayed" identifies elements that will not appear in the interface, and the 'Elements' ***group_name***, "Generic_Equation" identifies internally accessed equations that will also not appear in the interface.

A.7.1 Template of a 'Groups' Block

Groups
{

Fields
{

group_name
{
label = ***label***
This is the label that the user sees. The default is ***group_name***. This parameter is "language" sensitive!

short_label = ***label***
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***group_name***. This parameter is "language" sensitive!

} #(End of ***group_name*** Block)

} #(End of 'Fields' Block)

Elements
{

group_name
{
label = ***label***
This is the label that the user sees. The default is ***group_name***. This parameter is "language" sensitive!

short_label = ***label***
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***group_name***. This parameter is "language" sensitive!

} #(End of ***group_name*** Block)

} #(End of 'Elements' Block)

} #(End of Groups Block)

A.8 CrossRefs block

The 'CrossRefs' block is used to identify internal models used to determine winds or values from a given field or fields. Note that each ***wind_function_name*** or ***value_function_name*** refers to an FPA routine to calculate wind speed and direction or value based on the given field or fields in the cross-reference.

Note that ***element_name*** must be a member of the 'Elements' block, and that ***level_name*** must be a member of the 'Levels' block.

Note that ***unit_name*** must be a member of the 'Units' block.

A.8.1 Template of a 'CrossRefs' Block

CrossRefs
{

Winds
{

crossref_name
{
label = ***label***
This is the label that the user sees. The default is ***crossref_name***. This parameter is "language" sensitive!
short_label = ***label***
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***crossref_name***. This parameter is "language" sensitive!
description = ***label***
This is an optional descriptive label. This parameter is "language" sensitive!
wind_function = ***wind_function_name***
This parameter sets the name of a wind calculation function used to determine a given cross-reference wind. Note that each ***wind_function_name*** refers to an FPA routine to calculate wind speed and direction based on a given input field or fields. Preset choices are described in Table A.1. Note that user defined wind functions can also be used.
crossref_fields = None
OR
crossref_fields =
This parameter sets the field or fields used to determine the cross-reference wind from a given element and level.
{
element_name level_name
}
} #(End of ***crossref_name*** Block)

} #(End of 'Winds' Block)

Values
{
crossref_name
{
label = ***label***
This is the label that the user sees. The default is ***crossref_name***. This parameter is "language" sensitive!
short_label = ***label***
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***crossref_name***. This parameter is "language" sensitive!
description = ***label***
This is an optional descriptive label. This parameter is "language" sensitive!
value_function = ***value_function_name***
This parameter sets the name of a value calculation function used to determine a given cross-reference field. Note that each type of ***value_function_name*** refers to an FPA routine to calculate a value based on a given input field or fields. Preset choices are described in Table A.1. Note that user defined value functions can also be used.
crossref_fields = None
OR
crossref_fields =
This parameter sets the field or fields used to determine the cross-reference value from a given element and level.
{
element_name level_name
}

Note
The following keywords are used by the value calculation functions (referred to in Table A.1) **FPA_Daily_Max_Value_Func**, **FPA_Daily_Max_Time_Func**, **FPA_Daily_Min_Value_Func**, and **FPA_Daily_Min_Time_Func**.

---

time_weight = ***time_wgt unit_name***
value_weight = ***value_wgt unit_name***
In general, if a value is given for the normal time of the field, a maximum (or minimum) value at ***time_wgt*** away from the normal time must be more than ***value_wgt*** greater (or less) than the value at the normal time in order to be accepted as a new maximum (or minimum) value. This limits noise in the resulting fields by forcing the maximum (or minimum) values to the field closest to the normal time.
} #(End of ***crossref_name*** Block)
} #(End of 'Values' Block)

} #(End of 'CrossRefs' Block)

A.9 Samples block

The 'Samples' block is used to identify internal models used to sample parameters or winds from a given field. Note that each **value_samptype** or **wind_function** refers to an FPA routine to determine parameters or wind speed and direction based on a given field.

A.9.1 Template of a 'Samples' Block

Samples
{

Values
{

sample_type_name
{
label = ***label***
This is the label that the user sees. The default is ***sample_type_name***. This parameter is "language" sensitive!
short_label = ***label***
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***sample_type_name***. This parameter is "language" sensitive!
description = ***label***
This is an optional descriptive label. This parameter is "language" sensitive!
value_samptype =
This parameter sets the type of value to sample from a given field. Note that each type of value to sample refers to an FPA routine to determine parameters from the given field. Preset choices are described in Table A.1. Note that user defined sampling functions CANNOT be defined or used! Must be one of: **FPA_Sample_Value**, **FPA_Sample_Gradient**, **FPA_Sample_Curvature**, **FPA_Sample_Magnitude**, **FPA_Sample_Direction**
} #(End of ***sample_type_name*** Block)

} #(End of 'Values' Block)

Winds
{

sample_type_name
{
label = ***label***
This is the label that the user sees. The default is ***sample_type_name***. This parameter is "language" sensitive!
short_label = ***label***
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***sample_type_name***. This parameter is "language" sensitive!

description = *label*
This is an optional descriptive label. This parameter is "language" sensitive!

wind_function = *wind_function_name*
This parameter sets the name of a wind calculation function used to determine a wind sampled from the given field. Note that each ***wind_function_name*** refers to an FPA routine to calculate wind speed and direction based on the given field. Preset choices are described in Table A.1. Note that user defined wind functions can also be used.

} #(End of ***sample_type_name*** Block)

} #(End of 'Winds' Block)

} #(End of 'Samples' Block)

A.10 Sources block

The Sources block is used to identify information about data sources for FPA input and output fields. Some FPA data sources have sub-sources, that is, sub directories for the data. An example is the Donelan Lake Wave Model, which has subdirectories for each of the Great Lakes. These data sources are identified by ***source_name subsource_name***, but can also be identified by ***source_name:subsource_name*** if only a single string is to be used.

Note that ***element_name*** must be a member of the 'Elements' block, and that ***level_name*** must be a member of the 'Levels' block.

Note that ***crossref_name*** must be a member of the 'Winds' sub block of the 'CrossRefs' block, or the 'Values' sub block of the 'CrossRefs' block.

Note that ***unit_name*** must be a member of the 'Units' block.

Note that ***subfield_name*** must be a subfield of a plot object metafile!

A.10.1 Template of a 'Sources' Block

Sources
{

source_name
{

label = *label*
This is the label that the user sees. The default is source_name. This parameter is "language" sensitive!

short_label = *label*
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is source_name. This parameter is "language" sensitive!

description = *label*
This is an optional descriptive label. This parameter is "language" sensitive!

source_type =
This parameter sets the type of files for a data source. The choices are described in Table A.1. Must be one of: **Depiction**, **Guidance**, **Allied**, **Maps**, **Direct**, **NotUsed**.

minutes_required = True | False
This parameter identifies whether the data source uses minutes in the file timestamps. The default is False

alias = *alias_name(s)*
These can be shorter names for use in equations, for example. Alias names cannot be reused by another source. Only the first one encountered gets used.

directory_tag = None
OR

! directory_tag = *string*
This is a directory tag defined in the directories block of the setup file. (See Setup File Directory Override Options, (Appendix G) for more information.) Note that this parameter cannot be re-specified (unless changed from **None**)!

directory_path = None
OR

! directory_path = *string*
This is the directory path name in the file structure. This path is appended to the directory identified by directory_tag above. Note that this parameter cannot be re-specified (unless changed from **None**)!

directory_layers = *number_of_layers*
This specifies the number of data directories available for saving historical data. The default ***number_of_layers*** is 1, which would create just the data directory (specified by directory_tag and directory_path). A value greater than 1 will create (***number_of_layers*** - 1) `Prev` subdirectories. (For example, a value of 4 would create `base_directory/Prev/Prev/Prev`.) Note that increasing ***number_of_layers*** will automatically increase the number of subdirectories in the database, but decreasing ***number_of_layers*** will not delete subdirectories! The system rmdir command must be used to delete subdirectories in FPA databases.

subsources = None
OR

subsources =
These are subdirectories for each source_name. It is not necessary for a source to have subsources.
{
subsource_name
{
label = *label*
This is the label that the user sees. The default is ***subsource_name***. This parameter is "language" sensitive!
short_label = *label*
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***subsource_name***. This parameter is "language" sensitive!

description = *label*
This is an optional descriptive label. This parameter is "language" sensitive!

sub_directory_path = None
OR

! sub_directory_path = *string*
This is the subdirectory name in the file structure. This path is appended to the directory identified by directory_tag and directory_path above. Note that this parameter cannot be re-specified (unless changed from **None**)!

} #(End of ***subsource_name*** Block)

} #(End of subsources Block)

allied_model = None
OR

allied_model =
This parameter is used to identify information relating to an Allied Model. Each Allied Model in the FPA is identified by its own source_name, and information relating to the Allied Model is contained in this sub block. Note that the allied_model block MUST be specified if source_type = Allied. For all other source_type values, allied_model = None can be used. See Adding New Allied Models, (Appendix E) for more information on Allied Models.

{

time_matching = True | False
This parameter should be set to True if the Allied Model will be run from data files whose times will not match the times required by the Allied Model. The Allied Model will then attempt to find the closest available time to match the requested data. (For example, if the Allied Model requires hourly data and will be run from an FPA which will be interpolated to every second hour, then the Allied Model would use the same data for two hourly time steps.) This parameter should be set to False in all other cases.

source_info = *source_name* [*subsource_name*]
This parameter identifies the ***source_name*** (and ***subsource_name***, if necessary) of the default location of all input data files for the Allied Model. This is usually interp (for FPA interpolated data) or depict (for FPA depiction data).

pre_process = *pre_process_run_string*

process = *process_run_string*

post_process = *post_process_run_string*
These keywords set the run strings for the pre-processing, processing, and post-processing modules of a given Allied Model. These run strings can contain any of the following keywords, which automatically substitute the appropriate parameters into the run string when the Allied Model is run.
- **<SETUP>**: for the setup file name
- **<SOURCE>**: for the ***source_name*** and **<SUBSOURCE>**: for the ***subsource_name***
- **<RTIME>**: for the "zero" hour in the Depiction sequence
- **<ELEMENT>**: for the active depiction element
- **<LEVEL>**: for the active depiction level
- **<VTIME>**: for the active depiction

programs = None
OR
programs =
This parameter identifies the name(s) and location(s) of the actual Allied Model application(s) that will be launched by the FPA when the processing module of the Allied Model is run.
{
program_alias
{
directory_tag = None
OR
directory_tag = *string*
This is a directory tag defined in the directories block of the setup file. (See Setup File Directory Override Options, (Appendix G) for more information.) The default is the string from the directory_tag parameter declared at the start of the ***source_name*** section.
program_path = *string*
This is the application name in the file structure. This name is appended to the directory identified by directory_tag above.
} #(End of program_alias Block)
} #(End of programs Block)

files = None
OR
files =
This parameter identifies all input and output files required by the application or the FPA pre-processing, processing, or post-processing modules in running the Allied Model.
{
file_alias
{
directory_tag = None
OR
directory_tag = *string*
This is a directory tag defined in the directories block of the setup file. (See Setup File Directory Override Options, (Appendix G) for more information.) The default is the string from the directory_tag parameter declared at the start of the ***source_name*** section.
file_path = *string*
This is the file name in the file structure. This name is appended to the directory identified by directory_tag above.
} #(End of file_alias Block)
} #(End of files Block)

required_fields = None
OR
required_fields =
This parameter identifies all FPA fields required by the application to run the Allied Model. (These fields will be checked when the user requests the Allied Model to be run by the FPA.)
{
required_field_alias
{
field_info = ***element_name level_name*** **[** ***field_type*** **]**
This parameter identifies the required field by element and level. If ***field_type*** is specified as **Scattered** the labels of the required field are used.
sub_field_info = ***subfield_name unit_name***
This parameter identifies the sub field and 'units' of the sub-field for information from a plot object metafile. The ***subfield_name*** must be defined in the entity line in the metafile. (Can be used with **field_type = Scattered**)
source_info = ***source_name*** **[*****subsource_name*****]**
This parameter identifies the ***source_name*** (and ***subsource_name***, if necessary) of the location of the required field for the Allied Model. The default is the ***source_name*** [***subsource_name***] from the source_info parameter declared at the start of the allied_model block.
attribute_info_reset = True | False
This parameter resets the attribute_info list given below.
attribute_info = ***tag attrib_name attrib_units***
Multiple **attribute_info** entries can be included to build a list of attributes to use in the allied model. For attributes from Discrete, Line, Point and LChain type fields. The ***tag*** is an arbitrary name that will be used by the allied model to refer to each attribute. The ***attrib_name*** is the actual attribute (from the **attributes** list in the 'Elements' Block) which contains the value to be used, and the **attrib_units** gives the units of the attribute value. (Note that the use of tags instead of attribute names allows for more generic Allied Models.)
node_attribute_info_reset = True | False
This parameter resets the node_attribute_info list given below.
node_attribute_info = ***tag attrib_name attrib_units***
Multiple **node_attribute_info** entries can be included to build a list of node attributes to use in the allied model. For attributes from nodes on LChain type fields. The ***tag*** is an arbitrary name that will be used by the allied model to refer to each attribute. The ***node_attrib_name*** is the actual attribute (from the **attributes** list in the 'Elements' Block) which contains the value to be used, and the **node_attrib_units** gives the units of the attribute value. (Note that the use of tags instead of attribute names allows for more generic Allied Models.)
} #(End of required_field_alias Block)
} #(End of required_fields Block)

required_wind_crossrefs = None
OR
required_wind_crossrefs =
This parameter identifies all FPA wind cross-reference fields required by the application to run the Allied Model. (These wind cross-references will be checked when the user requests the Allied Model to be run by the FPA.)
{
required_wind_crossref_alias
{
crossref_info = *crossref_name*
This parameter identifies the required wind cross-reference. The ***crossref_name*** is from the Winds sub block of the 'CrossRefs' block.
source_info = *source_name* [*subsource_name*]
This parameter identifies the ***source_name*** (and ***subsource_name***, if necessary) of the location of the required wind cross-reference for the Allied Model. The default is the ***source_name*** [***subsource_name***] from the source_info parameter declared at the start of the allied_model block.
} #(End of required_wind_crossref_alias Block)
} #(End of required_wind_crossrefs Block)

required_value_crossrefs = None
OR
required_value_crossrefs =
This parameter identifies all FPA value cross-reference fields required by the application to execute the Allied Model. (These value cross-references will be checked when the user requests the Allied Model to be run by the FPA.)
{
required_value_crossref_alias
{
crossref_info = *crossref_name*
This parameter identifies the required value cross-reference. The ***crossref_name*** is from the Values sub block of the 'CrossRefs' block.
source_info = *source_name* [*subsource_name*]
This parameter identifies the ***source_name*** (and ***subsource_name***, if necessary) of the location of the required value cross-reference for the Allied Model. The default is the ***source_name*** [***subsource_name***] from the source_info parameter declared at the start of the allied_model block.
} #(End of required_value_crossref_alias Block)
} #(End of required_value_crossrefs Block)

metafiles = None
OR
metafiles =
This parameter identifies all FPA metafiles produced by running the Allied Model. The

metafiles are output to the directory determined from the directory_tag, directory_path, and sub_directory_path parameters identified above.

{

metafile_alias

{

file_alias = *file_alias*

This parameter identifies a member of the files block above from which the metafile information will be extracted.

field_info = *element_name level_name*

This parameter identifies the output metafile by element and level.

attribute_info_reset = True | False

This parameter resets the attribute_info list given below.

attribute_info = *tag attrib_name attrib_units*

Multiple **attribute_info** entries can be included to build a list of attributes to output from in the allied model. For attributes from Discrete, Line, Point and LChain type fields. The ***tag*** is an arbitrary name that will be used by the allied model to refer to each attribute. The ***attrib_name*** is the actual attribute (from the **attributes** list in the 'Elements' Block) which contains the value to be used, and the **attrib_units** gives the units of the attribute value. (Note that the use of tags instead of attribute names allows for more generic Allied Models.)

default_attrib_info_reset = True | False

This parameter resets the default_attrib_info list given below.

default_attrib_info = *attrib_name attrib_value*

Multiple **default_attrib_info** entries can be included to build a list of default attributes to output from the allied model. The ***attrib_name*** is the actual attribute (from the **attributes** list in the 'Elements' Block), and the **attrib_value** gives the default attribute value.

} #(End of metafile_alias Block)

} #(End of metafiles Block)

} #(End of allied_model Block)

} #(End of ***source_name*** Block)

} #(End of Sources Block)

A.11 Constants block

The 'Constants' block is used to identify meteorological values used in equations.

Note that ***unit_name*** must be a member of the 'Units' block.

A.11.1 Template of a 'Constants' Block

Constants

{

constant_name
{
label = *label*
This is the label that the user sees. The default is ***constant_name***. This parameter is "language" sensitive!
short_label = *label*
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***constant_name***. This parameter is "language" sensitive!
description = *label*
This is an optional descriptive label. This parameter is "language" sensitive!
constant = value *unit_name*
This is the numeric value and units for ***constant_name***.
} #(End of ***constant_name*** Block)

} #(End of Constants Block)

A.12 Units block

The 'Units' block is used to identify units that the FPA recognizes and the conversion of these units to and from MKS units.

Note that ***mks_unit_name*** must be a member of the 'Units' block.

A.12.1 Template of a 'Units' Block

Units
{

unit_name
{
label = *label*
This is the label that the user sees. The default is ***unit_name***. This parameter is "language" sensitive!
short_label = *label*
The short_label is used when there is not enough room on the FPA screen or button to use the longer label. The default is ***unit_name***. This parameter is "language" sensitive!
MKS_equivalent = *mks_unit_name*
This is an MKS unit that ***unit_name*** can be converted to.
MKS_conversion = *factor offset*
These are the parameters needed to convert ***unit_name*** to its MKS equivalent according to the equation:
MKS value = (value - offset)/factor.
} #(End of ***unit_name*** Block)

} #(End of 'Units' Block)

Appendix B

Equation Evaluator

The FPA contains an equation evaluator (or field calculator) to allow the user to create fields or sample point values based on equations that operate on fields within the FPA.

The equation evaluator is field based, that is, all variables in the equations must be fields that exist in the FPA data directories, or fields that can be calculated from fields that exist in the FPA data directories. Equations can be found in the "equation" blocks of the "Elements" section of the configuration files (as described in Config and Config.***name*** File Format, (Appendix A)).

B.1 Components of Equations

Equations are created from mathematical constants, functions, named constants or field names (and their modifiers) combined with each other by mathematical operators.

Mathematical constants are integer or real numbers in decimal or scientific notation, such as 535, -4.56, 89.7E5 or 1.22e-12. Functions are identified as a string followed by square brackets, that is, function[]. A function may or may not contain other fields within the square brackets. Fields within the brackets are separated by commas (,). Functions recognized within the FPA are described in the Equation Functions section below.

Named constants are strings found in the "Constants" section of the configuration files (as described in Config and Config.***name*** File Format, (Appendix A)). For example, the value of PI is defined in the `Config.master` configuration file as having a value of 3.1415926536. (Note that named constants are case sensitive, so that pi or Pi will not be recognized by the equation evaluator. Note also that named constants are evaluated before fields, so that if a configuration file uses the same name in the "Constants" section and the "Elements" section, then the equation evaluator will only recognize the "Constants" definition.)

Fields are usually defined by element names, which are strings found in the "Elements" section of the configuration files (as described in Config and Config.***name*** File Format, (Appendix A)). (Fields may also be defined by the combination of the element file identifier and the level file identifier used in the file naming convention in the FPA Data Directory structure. (See File Naming Conventions, (Section 4.3)) Element and level file identifiers can be found in the "Elements" and "Levels" sections of the configuration files.)

A field in the FPA is defined as an element at a given level for a given valid time that comes from a given source at a given run time. (A field in the FPA configuration files has a generalized definition given only by element and level.)

For example, the 850 mb dewpoint temperature for 12Z February 2, 2003 taken from the Canadian Global Environment Multiscale (GEM) model run at 00Z February 1, 2003 is a field. In the FPA parlance, the element is dewpoint, the level is 850mb, the valid time is 2003:033:12 (in FPA timestamp format of year: julian day: hour), the source is GEM, and the run time is 2003:032:00. All of this field information is required by the FPA to be able to find and display this field.

All equations in the FPA are defined for a given element (or occasionally a given element and level). This means that all other information about the field in the equation is implied. For example, the equation for dewpoint temperature in the `Config.master` configuration file is given as `temp - es`, or temperature minus dewpoint depression, which are both also FPA fields. Thus, when one requests the FPA to display the 850 mb dewpoint temperature for 12Z February 2, 2003 taken from the Canadian GEM Model run at 00Z February 1, 2003, the field is calculated from the temperature and dewpoint depression at 850 mb for 12Z February 2, 2003 taken from the Canadian GEM Model run at 00Z February 1, 2003. Information about the level, valid_time, source, and run_time is assumed to be the same as for the field that is being calculated, unless the equation explicitly modifies the information.

Modifiers for fields are identified by the field name followed by angle brackets, that is, name<modifier(s)> Modifiers within the brackets are strings separated by commas (,). The usage of field modifiers is discussed in the Field Modifiers section below.

Any field can have an equation which depends on fields that can also have equations. There is no limit to the number of equations which may be necessary to access to evaluate a given field. There is an internal check, however, to ensure that an infinite loop does not occur. (An example can be found in the `Config.master` configuration file, where dewpoint temperature has an equation requiring dewpoint depression which has an equation requiring dewpoint temperature! One of these fields would have to be present for the equation to succeed.)

The components of equations are summarized below:

numeric	constant value
name[]	function name
name	named constant (if found)
name	field name
name<modifier(s)>	field name with modifiers

The mathematical operators that can be used within the FPA equation evaluator are described in the Mathematical Operators section below.

B.2 Mathematical Operators and Their Precedence

Mathematical operators used inside FPA equations are a subset of HP C standard mathematical operators, with a few special additions.

The backslash character, that is \, is used for continuing a long equation onto another line. The effect of the backslash character in an equation string is to simply concatenate the two lines into one.

The @ and **!** characters are "magic" field or modifier substitution characters described in the Generic Equations section below.

The **$upper** and **$lower** strings are also "magic" layer to level modifiers described in the Field Modifiers section below.

The HP C standard left to right grouping of parameters is used, according to the operator precedence as given in the table below. Round brackets, that is **()**, can be used for grouping of parameters in equations for clarity, or to force evaluation of equations in the desired order.

The ^ character is used as a shorthand form of the power function. For example, **field** ^{2} would take the square of all values in field.

Note that the **+** and **-** signs used with mathematical constants can also be used as unary operators with functions, named constants, or fields. For example, **-field** would give the negative of field.

The mathematical operators used in equations are summarized below, in order of descending precedence:

operation	**meaning**
\	line continuation
@ !	"magic" for substitution in Generic Equations
$upper $lower	"magic" for layer levels in Field Modifiers
()	grouping for evaluation
+ -	unary plus or minus
^	power
* /	multiplication or division
+ -	addition or subtraction

The left to right grouping of parameters and the precedence of operators will affect the evaluation of any equation.

Example B.1 Order of operations

For example, the equation:

```
a + - b * c / d ^ e - f
```

would be evaluated as:

```
( a + ( ( ( -b ) * c ) / ( d ^ e ) ) ) - f
```

The user would have to use round brackets to force a different order of evaluation, as in:

```
a + ( ( -b ) * ( ( c  /  d )  ^  ( e  -  f ) ) )
```

Brackets can be nested for as many layers as is necessary to force the desired evaluation of the equation, or to clarify the order of evaluation. Long equations can be extended to additional lines with as many backslash characters as may be desired. The maximum length of an FPA equation in the configuration files is presently 1024 characters, including blank spaces.

B.3 Equation Functions

Mathematical functions within FPA equations are identified by a name followed by square brackets, as in **name[]**. The function may contain mathematical constants, other functions, named constants or field names

(and their modifiers) within the square brackets, with each parameter separated by the next by commas ',',. The number of parameters within the square brackets must match the number of parameters required by the given function. As well, the type of parameters, whether they are fields or values or either, is also important for some functions. For example, the x and y derivative functions must have a field as the parameter, since the derivative of a constant value is by definition, 0.

A number of HP C standard mathematical functions are also available. These functions are defined in the same format as functions defined within the FPA routines.

A list of functions for equations is summarized below, with the number and type of parameters required by each function (field, value, or either):

Table B.1: FPA equation editor functions

Function Name	Description	Type of each parameter
sin[***fld***]	sine of ***fld***	either
cos[***fld***]	cosine of ***fld***	either
tan[***fld***]	tangent of ***fld***	either
asin[***fld***]	inverse sine of ***fld***	either
acos[***fld***]	inverse cosine of ***fld***	either
atan[***fld***]	inverse tangent of ***fld***	either
atan2[***fld1***,***fld2***]	inverse tangent of (***fld1***/***fld2***)	either either
sinh[***fld***]	hyperbolic sine of ***fld***	either
cosh[***fld***]	hyperbolic consine of ***fld***	either
tanh[***fld***]	hyperbolic tangent of ***fld***	either
exp[***fld***]	exponential function e^***fld***	either
log[***fld***]	natural logarithm ln(***fld***)	either
log10[***fld***]	base 10 logarithm log(***fld***)	either
sqrt[***fld***]	square root of ***fld***	either
ceil[***fld***]	smallest integer not less than ***fld***	either
floor[***fld***]	largest integer not greater than ***fld***	either
fabs[***fld***] or abs[***fld***]	absolute value of ***fld***	either
fmod[***fld1***,***fld2***] or mod[***fld***,***fld2***]	floating point remainder of (***fld1***/***fld2***) with sign of ***fld1***	either either
copysign[***fld***]	sets sign of ***fld1*** from sign of ***fld2***	either either
hypot[***fld1***,***fld2***]	hypotenuse of ***fld1*** and ***fld2*** using sqrt(***fld1*******fld1*** + ***fld2*******fld2***)	either either
pow[***fld1***,***fld2***]	power by (***fld1***^***fld2***)	either either
plus[***fld1***,***fld2***]	addition by (***fld1*** + ***fld2***)	either either
minus[***fld1***,***fld2***]	subtraction by (***fld1*** - ***fld2***)	either either
mult[***fld1***,***fld2***]	multiplication by (***fld1*** * ***fld2***)	either either
divn[***fld1***,***fld2***]	division by (***fld1*** / ***fld2***)	either either
max[***fld1***,***fld2***]	maximum of ***fld1*** and ***fld2***	either either
min[***fld1***,***fld2***]	minimum of ***fld1*** and ***fld2***	either either
between[***fld***,***fld1***,***fld2***]	limit ***fld*** between ***fld1*** and ***fld2***	value value value or field either either
outside[***fld***,***fld1***,***fld2***]	limit ***fld*** outside ***fld1*** and ***fld2***	value value value or field either either

Table B.1: (continued)

Function Name	Description	Type of each parameter
lat[]	latitude	
lon[]	longitude	
ddx[***fld***]	derivative with respect to x of ***fld***	field
ddy[***fld***]	derivative with respect to y of ***fld***	field
curv[***fld***]	curvature of ***fld***	field
ddt[***fld***]	derivative with respect to time from ***fld*** at 2 to 5 sequential times	either
laplc[***fld***]	laplacian of ***fld*** using ddx(ddx(***fld***))+ddy(ddy(***fld***))	field
advct[***fld***,***fld1***,***fld2***]	advection of ***fld*** by u (***fld1***) and v (***fld2***) components of wind using -(***fld1****ddx(***fld***)+***fld2****ddy(***fld***))	field either either
divrg[***fld1***, ***fld2***]	divergence of u (***fld1***) and v (***fld2***) components of wind using ddx(***fld1***)+ddy(***fld2***)	field field
svprs[***fld***]	saturation vapour pressure from temperature ***fld***	either
lvlprs[]	pressure of a field level	
uprprs[]	pressure of the upper level of a field layer	
lwrprs[]	pressure of the lower level of a field layer	
sunang[]	solar zenith angle	
sundist[]	Earth to Sun distance	

B.4 Field Modifiers

A field in the FPA is defined as an element at a given level for a given valid time that comes from a given source at a given run time. If a given field has an equation, then the level, valid time, source, and run time for all fields in the equation are assumed to be the same as for field that is being calculated, unless these parameters are explicitly modified.

Modifiers for fields are identified by the field name followed by angle brackets, that is, name<***modifier(s)***> Modifiers within the brackets are strings separated by commas ',’.

The **$upper** and **$lower** modifiers are "magic" modifiers, to set the level of a field from the upper or lower levels of a layer.

Example B.2 field level modifier from a layer type field

An equation for an 850 to 1000 mb field containing the field **height<$upper>** would use the 850 mb height to evaluate the equation.

Modifiers which begin with @ or **!** are "magic" field or modifier substitutions described in the Generic Equations section below.

Modifiers are first checked against the list of known levels. If the modifier matches a level in the FPA configuration files, then the equation evaluator resets the level.

For all other modifiers, the first letter of the modifier string identifies the parameter that will be modified.

Modifiers which begin with **v** are used to reset valid time.

Modifiers which begin with **m** are used to reset source.

Modifiers which begin with **r** are used to reset run time.

Modifiers which begin with **u** or **l** are used to reset the upper or lower level for a layer type field.

Example B.3 level modifier with model and valid time

The modifier **height<850,mGEM,v-06>** resets level to 850mb, the source to the GEM model, and the valid time to 6 hours before the current valid time.

Values can also be extracted from the attribute of a discrete, wind, line or scattered field, as long as the attribute can be converted to a numeric value or use a lookup table to convert the attribute to a numeric value. The **a** modifier must be used to identify the attribute to be evaluated. The following are the modifiers that are used by attributes.

Modifiers which begin with **a** are used to set the field attribute.

Modifiers which begin with **b** are used to set the units for the field attribute (default is MKS).

Modifiers which begin with **x** are used to set the name of a lookup table for converting the field attribute value to a numeric value. The default directory for a lookup file is in `config/lookups`. The lookup file contains lines with a keyword to match against the extracted field attribute, and a numeric value to use if the match is successful. A line containing the keyword ***missing*** can be used to identify a numeric value to use if the field attribute is not found. A line containing the keyword ***default*** can be used to identify a numeric value to use if the field attribute does not match any of the keywords.

Modifiers that begin with **d** are used to set the default field attribute value (default 0.0).

Modifiers that begin with **p** are used to set the proximity to a feature for using the attribute value (default 0.0).

Modifiers that begin with **q** are used to set the units for proximity (default is km).

Table B.2: Field modifiers

Modifier	Meaning
@ !	"magic" for substitution in Generic Equations
level	reset level to ***level***
$upper or $lower	reset level from the upper or lower levels of a layer
u ***level***	reset upper level of a layer to ***level***
l ***level***	reset lower level of a layer to ***level***
v ***hrs***	reset valid time to run time plus number of ***hrs***
v+***hrs*** or v-***hrs***	reset valid time by plus or minus number of ***hrs***
v previous	reset valid time to next earliest time (if it exists)
v ***timestamp***	reset valid time to ***timestamp*** (in FPA timestamp format of year:julian day:hour or year:julian day:hour:minutes)

Table B.2: (continued)

Modifier	Meaning
m ***source*** or m ***source subsource***	reset source or source and subsource to ***source*** or ***source*** and ***subsource***
r+***hrs*** or r-***hrs***	reset run time by plus or minus number of ***hrs***
r current	reset run time to most recent run time of model
r previous	reset run time to next most recent run time of model
r ***timestamp***	reset run time to ***timestamp*** (in FPA timestamp format of year:julian day:hour or year:julian day:hour:minutes)
a ***field attribute***	set field attribute
b ***units***	set units for field attribute
x ***lookup***	set lookup table for field attribute conversion to numeric value
d ***value***	set default value for field attribute
p ***value***	set proximity value for field attribute to use
q ***units***	set units for proximity

B.5 Generic Equations

All equations in the FPA are "generic" in the sense that they can apply to more than one level, valid time, source, or run time. However, many fields have equations that are similar to those of other fields. For example, the geostrophic wind from a height field has the same equation as the thermal wind from a thickness field. Generic equations are used in the FPA equation handler to allow the same equation to be used for a number of similar fields. The FPA configuration files contain special fields for these generic equations which are strictly for calculation of other fields; they are not meant to be displayed or sampled. A special element group, Generic_Equation, has been defined in the Elements subblock of the Groups block of the `Config.master` configuration file (as described in Config and Config.***name*** File Format, (Appendix A)) for use with generic equations. Fields in this special group will not be displayed in any of the on-screen lists within the FPA.

Generic equations use special substitution characters to allow the equation to be used with more than one field. Field substitution is accomplished with the use of the @ character, and field modifier substitution with the use of the **!** character. The format of an equation using a generic equation is given by:

```
generic_equation < @fld,   !modf >
```

where ***fld*** is the field and ***modf*** the modifier to be substituted. The generic equation will refer to the field or modifier in its equation string by the characters @ and **!**. For example, the equation for GENERIC_difference, given by the equation

```
@  -  @ < ! >
```

would expand to

```
p - p <r-12>
```

if accessed by the generic equation string

```
GENERIC_difference < @p, !r-12 >
```

Table B.3: Generic equation substitution parameters

Substitution parameter	Meaning
@***fld***	substitute ***fld*** for @ in generic equations
@	use ***fld*** from @***fld*** in place of @ in equation
!***modf***	substitute ***modf*** for ! in generic equations
!	use ***modf*** from !***modf*** in place of ! in equation

B.6 Units of Equations

An equation in the FPA configuration files may be written in any convenient units, by simply setting the units in the "equation" block of the "Elements" section of the configuration file (as described in Config and Config.***name*** File Format, (Appendix A)).

However, it is important to remember that fields within FPA equations are ALWAYS evaluated in MKS units, as given by the MKS equivalent units corresponding to the units defined for the field in the "precision" line in the "Elements" section of the configuration file. For example, the msl pressure field is ALWAYS evaluated in units of Pa (Pascals), even though the units in the "precision" line are given as mb (millibars). Therefore, an equation containing the string pressure - 1000 will not give the difference between the msl pressure and 1000 mb; the string would have to be written as (pressure / 100) - 1000 to produce the desired result.

Appendix C

Presentation and Presentation.name

The `Presentation` and `Presentation.`***`name`*** files define the display information about all fields that FPA could potentially encounter. The `Presentation.`***`name`*** files are global `Presentation` files containing standard display information as well as information for specific FPA applications. The `Presentation` file is a "local" presentation file which will "include" one or more of the `Presentation.`***`name`*** files, as well as contain user-defined changes or additions to these files.

In this appendix when a parameter value must come from a predefined list, this list will be presented between '{' and '}' and the options are separated by the '|' character. Optional attributes are bracketed between the '[' and ']' characters. Comments are presented between '(' and ')'. The "appropriate display options" mentioned in the summary below specify how the objects are to be displayed. Since many of the display options are common amongst the different member types they are described once at the end of this appendix.

C.1 Presentation Format Summary

The following represents the basic format of a presentation file entry.

```
revision revision_number

 field element level source

  member {continuous | vector} dname
    units uname
    contour range min max standard increment
      (appropriate display options)
    contour list val val ...
      (appropriate display options)
    vector multiplier (vector only)
      (appropriate display options)
    maxima min max
      (appropriate display options)
    minima min max
      (appropriate display options)
```

```
    saddle    min max
      (appropriate display options)

  member {nodes | spot} dname
    class cname
      class_member {barb | button | label | mark} mname attribute
        category value
          (appropriate display options)
        attribute name value
          (appropriate display options)
        default
          (appropriate display options)

  member {area| barb| button| curve| discrete| label| lchain| mark} dname
    category value
      (appropriate display options)
    attribute name value
      (appropriate display options)
    default
      (appropriate display options)

  member plot dname
    subfield {barb | button | label | mark} sname
      (appropriate display options)

include include_file
```

Most blocks (shown by indentation here) can be repeated as many times as needed. However, for some blocks (vector, maxima, minima, saddle, default) it doesn't make sense to have more than one per member or class_member block. The following section expands this basic format and explains the options and requirements for each block.

C.2 Format Explanation

Now that you have the basic idea of what a presentation file entry looks like let's take a closer look at each block.

revision *revision_number*
: The revision line must be the first uncommented line in each presentation file. (The current revision is 8.0)

field *element level source*
: Introduces presentation information for the given field. All keywords which follow, up to the next field line, describe how this field is to be presented. The parameters are:

  - element (may be ALL)
  - level (may be ALL)

- source (may be ALL)

member {continuous | vector} ***dname***

Introduces presentation information for the given member of the current field. All keywords which follow, up to the next field or member line, describe how this member is to be presented. The parameters are:

- The field type of this member, for this example it is one of continuous or vector.
- Arbitrary (descriptive) name for this member

units ***uname***

Continuous and Vector fields are usually represented by a numerical value. This keyword defines the conversion from MKS to desired units for this particular field member. The parameter is:

- unit name (as defined in Config)

contour range ***min max standard increment***

Defines a set of contours to be generated for the current field member and introduces presentation information for these contours. The parameters are:

- lowest contour value (* implies no minimum)
- highest contour value (* implies no maximum)
- standard contour value (increment counts from this value)
- contour value increment

Appropriate display options are: colour, scale, style and width

contour list ***val val ...***

Defines a set of contours to be generated for the current field member and introduces presentation information for these contours. The parameters are:

- a list of specific contour values.

Appropriate display options are: colour, scale, style and width.

vector ***multiplier*** **(vector only)**

Defines a grid of barbs or arrows to display with a vector field. The parameter is:

- multiplier: This is an integer value. The usual value is 1, which will display a barb/arrow spaced according to the resolution of the FPA base map. A value greater than 1 will display that number of barbs/arrows for each map grid. A value less than 1 will skip that number of map grids.

Appropriate display options are:btype, bvaloff, bcolour, colour, font, length, scale and width.

maxima ***min max***

Defines whether local maxima are to be displayed, and introduces presentation information for the maxima displayed. The parameters are:

- lowest value (* implies no minimum)
- highest value (* implies no maximum)

Appropriate display options are: angle, colour, font, hjust, vjust, marker, offset, scale, shadow, size and symbol

minima ***min max***

Defines whether local minima are to be displayed, and introduces presentation information for the minima displayed. The parameters are:

- lowest value (* implies no minimum)
- highest value (* implies no maximum)

Appropriate display options are: angle, colour, font, hjust, vjust, marker, offset, scale, shadow, size and symbol

saddle ***min max***

Defines whether local saddle (col) points are to be displayed, and introduces presentation information for the saddle points displayed. The parameters are:

- lowest value (* implies no minimum)
- highest value (* implies no maximum)

Appropriate display options are: angle, colour, font, hjust, vjust, marker, offset, scale, shadow, size and symbol

member {nodes | spot} ***dname***

Introduces presentation information for the given member of the current field. All keywords which follow, up to the next field or member line, describe how this member is to be presented. The parameters are:

- The field type of this member, for this example it is one of nodes or spot.
- Arbitrary (descriptive) name for this member.

class ***cname***

Introduces presentation information for the given class of spot objects from the current field member. Consider a "spot" object to be a complex label, made up of a group of simpler member objects, clustered around the evaluation point (similar to a "station plot"). Spot objects of the same class have the same layout, as described by subsequent class_member keywords. The value of class corresponds to the value of type_class specified in the scattered types section of your config entry for this field member. The parameter is:

- class name

class_member {barb | button | label | mark} ***mname*** **[*attribute*]**

Defines a new member object for the current class of spot objects, and introduces presentation information for that member. The parameters are:

- member type is one of:
  - barb - displays subfield as a wind barb or arrow,
  - button - displays subfield as a button,
  - label - displays subfield as a text label,
  - mark - displays subfield as a marker or text symbol.
- member name
- attribute name (This is the attribute for the label display).

category ***value***

Introduces presentation information for objects in the current spot class member that match the given category. The parameter is:

- The category value corresponds to an FPA_category for this field.

If you specified an attribute name for the class_memeber above the default label text will be the value of that attribute otherwise it will be the FPA_category value.

Appropriate display options are: angle, btype, bvaloff, bcolour, colour, font, hjust, vjust, length, marker, offset, scale, shadow, size and symbol.

attribute *name value*

Introduces presentation information for objects in the current spot class member that match the given value of the given attribute. The parameters are:

- attribute name
- attribute value (This is the value to match with the value from attribute name.)

If you specified an attribute name for the class_memeber above the default label text will be the value of that attribute otherwise it will be the value of the attribute you are grouping by.
Appropriate display options are: angle, btype, bvaloff, bcolour, colour, font, hjust, vjust, length, marker, offset, scale, shadow, size and symbol.

default

Introduces presentation information for objects in the current spot class member that do not match any of the category or attribute selections introduced for the same field member. No parameters. You must specify an attribute name for the class_member if you want labels. The default label text will be the value of that attribute.
Appropriate display options are: angle, btype, bvaloff, bcolour, colour, font, hjust, vjust, length, marker, offset, scale, shadow, size and symbol.

member {area| barb| button| curve| discrete| label| lchain| mark} *dname*

Introduces presentation information for the given member of the current field. All keywords which follow, up to the next field or member line, describe how this member is to be presented. The parameters are:

- The field type of this member, for this example it is one of area, barb, button, curve, discrete, label, lchain or mark.
- Arbitrary (descriptive) name for this member.

category *value*

Introduces presentation information for objects in the current field member that match the given category. The parameter is:

- The category name corresponds to a possible value of FPA_category for this field

Appropriate display options are: angle, btype, bvaloff, bcolour, colour, fill, font, hatch, hjust, vjust, length, marker, offset, pattern, scale, shadow, size, style, symbol and width.

attribute *name value*

Introduces presentation information for objects in the current field member that match the given value of the given attribute. The parameters are:

- attribute name
- attribute value (This is the value to match with the value from attribute name.)

Appropriate display options are: angle, btype, bvaloff, bcolour, colour, fill, font, hatch, hjust, vjust, length, marker, offset, pattern, scale, shadow, size, style, symbol and width.

default

Introduces presentation information for objects in the current field member that do not match any of the category or attribute selections introduced for the same field member. No parameters.

Appropriate display options are: angle, btype, bvaloff, bcolour, colour, fill, font, hatch, hjust, vjust, length, marker, offset, pattern, scale, shadow, size, style, symbol and width.

member plot *dname*

Introduces presentation information for the given member of the current field. All keywords which follow, up to the next field or member line, describe how this member is to be presented. The parameters are:

- The field type of this member, for this example it is plot.
- Arbitrary (descriptive) name for this member.

subfield {barb | button | label | mark} *sname*

Defines a new subfield for the current plot field member, and introduces presentation information for that subfield. The parameters are:

- subfield type is one of:
  - barb - displays subfield as a wind barb or arrow,
  - button - displays subfield as a button,
  - label - displays subfield as a text label,
  - mark - displays subfield as a marker or text symbol.
- subfield name

Appropriate display options are: angle, btype, bvaloff, bcolour, colour, font, hjust, vjust, length, marker, offset, scale, shadow, size, symbol.

include *include_file*

The include lines allow several presentation files to be used at the same time. The presentation file reader adds the information from the "included" presentation files to the information in the current presentation file to determine the presentation of all fields encountered. Omitted attributes resort to default values established in the software which reads the presentation file. (for example: colour = Black, style = solid, font = simplex, etc).

Note

Note: Unlike the configuration file reader the presentation file reader uses the first matching entry it encounters. That is, if a given member of a given field is already described in a Presentation Configuration File included at the end of the file, then the information in the file overrides the default appearance for that member only. The default appearance of other members is not affected. If the member is not already described, then this new information serves to add a new member to the given field, and defines its appearance.

C.3 Display Options

The following keywords refer to the objects selected from the current field member, spot class member, or plot subfield, specified by one or more of the above keyword specifications, and describe specifically how those objects are to be displayed:

Table C.1: Display option descriptions

Option	Explanation
angle	Defines the orientation of a label or mark.The parameter is: • floating point angle of rotation in degrees.
btype	Defines the type of wind or vector field display. The parameters are: • type: This can be one of wind (normal wind barb), arrow (draw as an arrow) or none (do not display). • tofrom: This can be one of dirto (direction defines where flow is going), or dirfrom (direction define where the flow comes from). The parameters to, dir_to, from and dir_from are also valid. • doval: This can be either showval (include the magnitude as a label). or noval (no label). The parameters showvalue or novalue are also valid. • units for wind or vector field (as defined in Config)
bvaloff	Defines the offset of a wind/wave magnitude label from the given position (see also btype). The parameters are: • x offset: Horizontal offset in map or screen units. (see scale) • y offset: Vertical offset in map or screen units.
bcolour	Defines the colour for a wind/wave vector and its optional magnitude label (see also btype). The parameters are: • barb colour: See colour for supported values. • label colour: See colour for supported values.
colour	Defines text, mark or line colour. The parameter is: • colour - Colours may be identified by one of: – X colour name (e.g. SlateBlue), – X hexadecimal RGB colour notation (#rrrgggbbb) or – a direct pixel value (=0 through =255)

Table C.1: (continued)

Option	Explanation
fill	Defines the appearance of filled objects. The parameters are: • fill colour: See colour for supported values. (Fill type defaults to solid_fill) or • fill type: Supported fill types are: hollow_fill, solid_fill, hatch_fill • fill colour: See colour for supported values
font	Defines the font for text. The parameter is: • font name: Supported font names are: simplex, bold, helvetica, helvetica_bold, times or serif, times_bold or bold_serif, fpasymbol-24. The default is simplex.
hatch	Defines hatch fill attributes when fill has been set to hatch_fill. The parameters are: • hatch type: Supported hatch types are: parallel_hatch and cross_hatch. The default is parallel_hatch. • hatch space: See size for supported values • hatch angle: This is a real angle between 0 and 360 degrees.
hjust	Defines the horizontal justification for labels. The parameters are: • mode: This may be one of right(r), centre(c) or left(l). The default is centre(c)
vjust	Defines the vertical justification for labels. The parameters are: • mode: This may be one of Top (top of character cell), top(t), centre(c), bottom(b) or Bottom (bottom of character cell).The default is centre(c)
length	Defines the length of certain objects (e.g. wind barb) or a pattern repeat length. The parameter is: • item length: Length in map or screen units (see scale).
marker	Defines the marker type to be used for mark objects. The parameter is: • marker type: Supported marker types are: dot, plus, asterisk, circle, ellipse, cross, triangle, square, rectangle, diamond, circle_cross, square_cross, circle_fill, square_fill, rectangle_fill, triangle_fill, diamond_fill, circle_target. The default is dot.

Table C.1: (continued)

Option	Explanation
offset	Defines the offset of a plot sub-field or spot member from the plot centre. The parameters are: • x offset: Horizontal offset in map or screen units (see scale) • y offset: Vertical offset in map or screen units.
pattern	Defines patterned lines. The parameter is: • pattern file: This is the name of a pattern file which must exist in the patterns directory.
scale	Defines the interpretation of various sizes and dimensions. The parameter is: • mode: Supported scale modes are: window or device or vdc (0) for units with respect to the screen or map or geog or geography (1) for units with respect to the FPA map. The default is map.
shadow	Defines top and bottom shadow colours for text or marks. The parameters are: • top colour: See colour for supported values. • bottom colour: See colour for supported values.
size	Defines the size of text or marks. The parameter is: • item size: Height in map or screen units (see scale) or one of: tiny(5), small(10), label (usual label size is 20), legend (usual legend size is 25), medium(30), hilo(usual high/low symbol size is 40), large(50), huge(70), giant or gigantic or enormous or humongous(90).
style	Defines the line style for curves and area boundaries. The parameter is: • line style: Supported line styles are: solid, wide (default with width parameter set), dash, dashed, dot, dotted, dash_dot. The default is solid.
symbol	Defines a text symbol to be used for marks rather than a marker type. The parameter is: • string: This can be any printable ASCII string including extended ASCII.
width	Defines line width or pattern amplitude. The parameter is: • line width: real dimension in map or screen units (see scale) or one of zero or normal or thin(0), medium(3), thick(6), fat(9).

C.4 Examples

The following examples were drawn from $FPA/config/Presentation.master.

Example C.1 Vector & Contour

Describes the presentation for the wave height vector field for all levels and all models. There are dashed yellow contours at 1 and 2 meters, solid yellow contours at 3 and 4 meters and solid red contours every 2m starting from 5m. A grid of thin yellow arrows will be displayed with one arrow per map element. Maxima will be denoted by yellow plus marks and minima will be denoted by yellow diamond marks

```
field    uv_wave_hgt    ALL    ALL
  member    vector
    units     m
    contour   list 1 2
      colour  Yellow
      style   dash
    contour   list 3 4
      colour  Yellow
      style   solid
      width   1
    contour   range 5 * 5 2
      colour  Red
      style   solid
      width   1
    vector    1
      colour  Yellow
      btype   arrow    dirto    noval
      width   thin
      length  30.0
    maxima    1 *
      colour  Yellow
      marker  plus
    minima    1 *
      colour  Yellow
      marker  diamond
```

Example C.2 Plot

Describes the presentation for the storm plot field with level msl and for all models. Each point will feature a circle with a cross surround by value 1 above, value 2 to the left and value 3 to the right. The values appear in Green Simplex font and the marks are DeepPink.

```
field    storms    msl    ALL
  member  plot
    subfield  label value1
      offset  0 +10
      colour  Green
```

```
      font    simplex
      size    15.0
    subfield  label value2
      offset  -10 0
      colour  Green
      font    simplex
      size    15.0
    subfield  label value3
      offset  10 0
      colour  Green
      font    simplex
      size    15.0
    subfield  mark c1
      colour  DeepPink
      marker  circle
      size    10.0
    subfield  mark c2
      colour  DeepPink
      marker  cross
      size    7.0
```

Example C.3 Discrete with Labels

Describes the presentation of the weather_system area field for all levels and all models. In this example presentation information has been grouped by FPA_category. The categories are freezing, frozen, precip, vis, and cloud. The outline colour, line style, and line width are set for each category. The presentation for weather labels is also set here. In addition to the previous categories we have a category none and a default presentation blocks. The label size, colour, shadow, and font are set.

```
field weather_system ALL ALL
  member discrete weather
    category   freezing
      colour   Cyan
      style    solid
      width    5
    category   frozen
      colour   SkyBlue
      style    solid
      width    3
    category   precip
      colour   Green
      style    solid
      width    3
    category   vis
      colour   Yellow
      style    solid
      width    2
    category   cloud
      colour   Firebrick
```

```
    style    solid
    width    1
member spot labels
  class area
    class_member label value
      category  none
        size    label
        colour  Cyan
        shadow  CornflowerBlue CornflowerBlue
        font    simplex
      category  frozen
        size    label
        colour  Black
        shadow   SkyBlue SkyBlue
        font    simplex
      category  freezing
        size    label
        colour  Black
        shadow  Cyan Cyan
        font    simplex
      category  precip
        size    label
        colour  Green
        shadow  Firebrick Firebrick
        font    simplex
      category  vis
        size    label
        colour  Yellow
        shadow  Firebrick Firebrick
        font    simplex
      category  cloud
        size    label
        colour  Goldenrod
        shadow  Firebrick Firebrick
        font    simplex
      default
        size    label
        colour  Green
        font    simplex
```

Appendix D

Product Definition File Formats

D.1 Graphics Product Generator (GPGen)

The Graphics Product Generator application allows users to create tailored graphical products or simpler ASCII products from the FPA database. The two graphical product applications are PSMet (producing output in PostScript format) and SVGMet (producing output in SVG format, an open XML format). The ASCII application is TexMet. All three applications use a command language in product generation files to sample the FPA database and display the results in a number of different formats. Please refer to the **`Graphics Product Generator Reference Manual`** for details on the functionality of the applications and their product definition files.

D.2 FPA Metafile Copy

This file defines the FPA metafiles to be copied. The usual use of this facility is to send a set of FPA metafiles to another location where they can be used as initial fields or as guidance. The parameters are:

source
: The source of the metafiles. This is normally **`depict`** or **`interp`**.

time
: The time of the depictions. This is either **`ALL`** or the time of the depictions with respect to T0. This can be one of two formats: the time offset from T0 or the day relative to T0 and an absolute hour. For example, 0 6 12 24 would be taken as T0, T+6, T+12 and T+24. 0/12 1/0 1/6 would be taken as 12Z on the same day as T0, and 00Z and 06Z on the day after T0.

fields
: One of **`ALL`** or a list of **`element level`** pairs defining the required fields. For example: pressure msl temperature sfc ...

send_link_data
: **`True`** or **`False`**. Send the time link data. Default False.

target_directory
The directory into which to copy the metafiles. If the directory does not exist it is created. If target_directory is not specified a temporary directory is created. This temporary directory and all of its contents is removed after the post process procedure is executed.

post_process
Shell commands which will be executed once the metafiles are copied to the target directory. This can be used, for example, to ftp the files to another location. If a target directory was not specified above, then the temporary directory into which the requested depictions are placed is referenced by the macro '$METADIR' in the post process lines. Long commands can be broken up into multiple lines by terminating a line with a backslash '\'.

Example D.1 Graphics Metafile copy example

The following example would send all fields from four depictions to a remote site using the script ftp_script (your script):

```
source       = depict
time         = 0/12 0/18/ 1/0 1/12
fields       = ALL
post_process = tar -cf /tmp/meta.tar $METADIR; compress /tmp/meta.tar; \
                  ftp_script office1 /tmp/meta.tar; rm /tmp/meta.tar
```

Appendix E

Adding New Allied Models

Allied Models are applications or programs which can be run from the FPA, but are not actually part of the FPA. Allied Model processing is accomplished by a series of specialized modules;

- an initialization module (if required) which creates constant files required by the application,
- a pre-processing module which translates FPA data into a form usable by the particular application,
- an execution module which runs the application,
- and a post-processing module which translates output from the application into FPA type data files which can be imported back into the FPA depiction sequence.

Note that only the specialized modules make extensive use of the FPA Standard Library Routines. The Allied Models themselves are meant to be treated much like a "black box"; only minimal changes are required to run an Allied Model in the FPA environment.

Adding a new Allied Model requires knowledge of the Local Setup file (Defining Your Local Setup, (Chapter 5)) and the Local Configuration file (Config and Config.***name*** File Format, (Appendix A)).

Developing the specialized modules will also require access to the FPA Standard Library routines. The FPA Standard Library is included with the latest FPA distribution. Sample code located in `$FPA/templates/userinput` and `$FPA/templates/useroutput` can be used as a starting point for writing your own pre-process and post-process applications.

The last three sections of this Appendix describe the "FPA Warp", "FPA Create Area" and "FPA Create Contour" Allied Models. The "FPA Warp" allied model can be used to merge point data with Continuous fields. The "FPA Create Area" allied model can be used to generate proximity envelopes around features in Line, Link Chain or Scattered type fields. The "FPA Create Contour" allied model can be used to generate Discrete areas from the contours of Continuous fields. These Allied Models are available to be used by any user without coding changes.

E.1 Development Directories and FPA Integration

We recommend that the user create a series of development directories for implementation of any new Allied Model. The following directories are required:

- The user should create a Development Directory somewhere on the system, such as `your_directory/AModels.TEST`. The location of this Development Directory (identified by the keyword 'AModels.Test') must be added to the 'directories' block of the Local Setup file, as described below. Code and executables for the application itself, as well as constant input data files used by the application, will reside on specified subdirectories of `your_directory/AModels.TEST`.

- Variable input data files and all output files used by the application, as well as all FPA type data files from post-processing, will be located in the Data Directory `/home/data/AModels.DATA`, whose location is identified by the keyword 'AModels.Data' in the 'directories' block of the Local Setup file. These files will reside on specified subdirectories that the user may have to create, though the base Data Directory and all subdirectories are usually created automatically when the Allied Model is run.

- The code for the specialized initialization, pre-processing, execution, and post-processing modules for new Allied Models can reside on a Processing Directory, such as ***your_processing_directory***, which can be located anywhere on the system, as long as the path to the executable code is located in the users path. (The executables for Allied Models currently implemented in the FPA are located in the directory `$FPA/bin`.)

The directory structure used by a new Allied Model is outlined below. In this example, the new Allied Model is NewModel which can be run over two subareas (or subsources), Area1 or Area2. The Source Code and Executable Code refer to the actual Allied Model program or application.

Example E.1 Development Directory

```
ls your_directory/AModels.TEST/NewModel/
bin src Area1 Area2
```

where:

- `bin` contains executable code
- `src` contains source code
- `Area1` contains constant files
- `Area2` contains constant files

Example E.2 Data Directory

```
ls /home/data/AModels.DATA/NewModel
Area1 Area2
```

where:

- `Area1` contains variable files.
- `Area2` contains variable files.

Example E.3 Processing Directory

```
ls your_processing_directory/NewModel
Makefile init.c preproc.c alliedm.c postproc.c
```

where:

- `Makefile` is a makefile which accesses the FPA Standard Library Routines
- `init.c` is your initialization code
- `preproc.c` is your pre-processing code
- `alliedm.c` is your allied model code
- `postproc.c` is your post-processing code

The user should create a database for testing the new Allied Model using the command mkfpadb, and following the customization instructions. This procedure will also create the user's Local Setup file (***local_setup***).

The user will have to create all other directories required by the new Allied Model with the system mkdir command. The user would create the Development Directory by commands such as:

```
mkdir -p your_directory/AModels.TEST/NewModel/bin
mkdir -p your_directory/AModels.TEST/NewModel/Area1
```

The Data Directory can contain data from several successive runs of the new Allied Model. The 'run time' for the Allied Model will be written to the directory in a file called 'Dstamp', with the format 'Year:JulianDay:Hour' (for example, 1995:305:12). All files from the Allied Model run at any earlier time are automatically shuffled down to a Prev subdirectory, if one exists. (Note, however, that the Data Directory will be created automatically, if required, when the new Allied Model is run. The number of Prev subdirectories is determined by the 'directory_layers' line in the Local Configuration file described below.)

The user could create a Data Directory with the ability to save three successive runs of the new Allied Model with the command:

```
mkdir -p home/data/AModels.DATA/NewModel/Area1/Prev/Prev
```

The user will also have to modify the following files to integrate the new Allied Model into the FPA environment:

- The Local Setup file (***local_setup***)
- The Local Configuration file (***local_config***)

The Local Setup file and Local Configuration file are created when the user creates a new database. The user will have already customized the Local Setup file when creating the database. Further additions to the Local Setup file and Local Configuration file are described below.

E.2 The Local Setup File

The format of the Local Setup file is described in Defining Your Local Setup, (Chapter 5). The location of directories required by the new Allied Model is in the 'directories' block of the Local Setup file.

Note
The locations of all directories used by the new Allied Model should be defined in the Local Setup file, and not in the Allied Model processing programs.

The location of the Development Directory for the application itself and constant input data files must be added in 'directories' block, as in:

```
amodels.test your_directory/AModels.TEST
```

Note
The entry for 'amodels.exec' in the 'directories' block refers to the location of other Allied Model applications already integrated into the FPA.

Note
The entries for 'home' and 'Data' in the 'directories' block refer to the location of data files for the FPA, and that the entry for 'AModels.Data' in the 'directories' block refers to the usual location of the Data Directory for Allied Model input and output files. The path name of this directory is thus given by the path names of these three parameters, as `/home/Data/AModels.Data`. The subdirectories and files for the new Allied Model are described in the Local Configuration file section below.

The Allied Model to be run must also be added to the "[allied.model]" section of the 'interface' block. Each uncommented line in this section identifies an application which will appear in the Allied Model menus of the FPA as an application that can be run from within the FPA.

An Allied Model which can be run at several different locations can appear more than once, as in:

```
NewModel Area1
NewModel Area2
```

E.3 The Local Configuration File

The format of the Local Configuration file is described in Config and Config.*name* File Format, (Appendix A). The location of this file is given in the 'directories' block of the Local Setup file by the keyword 'config'. The name of the Local Configuration file is given in the 'config_files' block of the Local Setup file by the keyword 'config'. (The usual name of the Local Configuration file is 'Config'.)

The user must add another 'Sources' block to the Local Configuration file to describe the new Allied Model directory and file locations, the data files required from the FPA to run the Allied Model, and the data files

that the Allied Model can produce for the FPA to use. The format of the 'Sources' block is given in Config and Config.*name* File Format, (Appendix A). An example of a 'Sources' block for an Allied Model can be found in the file `$FPA/templates/useroutput/README`

```
Sources
  {
  NewModel
    {
    label            = "New Allied Model"
    short_label      = "New Model"
    description      = "New Allied Model Example - FPA Version 6"
    source_type      = Allied
    directory_tag    = AModels.Data
    directory_path   = NewModel
    directory_layers = 3 # number of directory layers
    subsources       =
      {
      Area1
        {
        label              = "New Model Area1"
        short_label        = "Area1"
        sub_directory_path = Area1
        }
      } # End of subsources Block
```

Note that the 'directory_tag' line identifies the default data directory keyword from the 'directories' block of the Local Setup file, and the 'directory_path' line identifies the default subdirectory for the data files. Any 'directory_tag' or '..._path' lines in the following sections override these default values. Note that the pathname in the 'sub_directory_path' line is appended to each subdirectory for each 'subsource' of the Allied Model.

Note that the 'directory_layers' line identifies the number of directories available for saving data from successive runs of the Allied Model. For example, 'directory_layers = 3' would result in a base data directory and two Prev subdirectories.

The 'allied_model' section below is used to identify all Allied Model parameters. The 'source_info' line identifies the default location for all FPA input data fields. Run strings for the 'pre_process', 'process', and 'post_process' modules can include any of the following keywords, which automatically substitute the appropriate parameters into the run string when the Allied Model is run.

Table E.1: Allied Model process keywords

Keyword	**Meaning**
<SETUP>	the Local Setup file (local_setup)
<SOURCE>	the Allied Model name in the Sources Block (NewModel)
<SUBSOURCE>	the Allied Model subarea in the sub_sources Block (Area1, for example)
<RTIME>	'zero' hour in the Depiction sequence, that is, the time at which the application is run (rtime) (Note that this is the time that is written to the 'Dstamp' file in the Data Directory when the application is run)
<ELEMENT>	The active element in the depiction editor

Table E.1: (continued)

Keyword	Meaning
<LEVEL>	The active level in the depiction editor
<VTIME>	The active depiction time

```
allied_model    =
  {
  time_matching = False
  source_info   = interp or depict
  pre_process   = allied_prep arg ... # run string for pre-processing
  process       = allied_exec arg ... # run string for execution
  post_process  = allied_post arg ... # run string for post-processing
```

The 'programs' section below is used to identify the actual Allied Model application that will be run when the specialized execution module is run. The ***program_alias*** will be used in the specialized execution module code to identify the Allied Model program.

```
  programs          =
    {
    program_alias
      {
      directory_tag = AModels.Test
      program_path  = bin/executable
      }
    } # End of programs Block
```

The 'files' section below is used to identify all input or output files used by the actual Allied Model application. The location of the files will default to the 'directory_tag' given at the start of the 'Sources' declaration (that is, AModels.Data) unless the 'directory_tag' is reset for an individual file. Note that ***file_alias_name*** will be used in the specialized module code to identify a file.

```
  files           =
    {
    file_alias_name # For Constant Files
      {
      directory_tag = AModels.Test
      file_path     = file
      }
    file_alias_name # For Variable Files
      {
      file_path     = file
      }
    } # End of files Block
```

The 'required_fields', 'required_wind_crossrefs', and 'required_value_crossrefs' sections below are used to identify the input FPA data used by the pre-processing module of the Allied Model. If not required, these can be set to None. The location of this input data will default to the 'source_info' declaration given at the start of the 'allied_model' section unless it is reset here.

Note
The ***required_field_alias*** or the ***required_crossref_alias*** will be used in the pre-processing module code to identify all required data.
The ***tag_label*** names will be used in the pre-processing module code to extract attribute information from the input data.

The ***element*** and ***level*** names in the 'field_info' declarations, the ***source*** names in the 'source_info' declarations, and the ***crossref*** names in the 'crossref_info' declarations must be declared in the 'Elements', 'Levels', 'Sources', or 'CrossRefs' blocks of a Configuration file! The user will have to include declarations in the Local Configuration file for any ***element***, ***level***, ***source***, or ***crossref*** data which is required by a new Allied Model but is not already declared. The user can refer to Config and Config.***name*** File Format, (Appendix A), as well as the examples given in the `Config.name` files in `$FPA/config`.

```
required_fields          =
  {
  required_field_alias
    {
    # input data identified by element and level
    field_info               = element  level
    # optional: member of a Plot Object
    sub_field_info           = sub_field
    # optional: location of input field
    source_info              = source
    # optional: reset list of attributes
    attribute_info_reset     = True | False
    # optional: attributes required by allied model
    # (may be specified more than once to generate a list)
    attribute_info           = tag_label  attrib_name  attrib_units
    # optional: reset list of link chain node attributes
    node_attribute_info_reset = True | False
    # optional: link chain node attributes required by allied model
    # (may be specified more than once to generate a list)
    node_attribute_info      = tag_label  attrib_name  attrib_units
    }
  } # End of required_fields Block
required_wind_crossrefs  =
# OR
required_value_crossrefs =
  {
  required_crossref_alias
    {
    # input data identified by cross-reference
    crossref_info        = crossref
    # optional: location of input field
    source_info          = source
    }
  } # End of required_..._crossrefs Block
```

The 'metafiles' section below is used to identify the output FPA data produced by the post-processing module of the Allied Model. The location of the files will default to the 'directory_tag' given at the start of the 'Sources' declaration (that is, AModels.Data).

Note

The ***metafile_alias*** will be used in the post-processing module code to identify all metafiles.
The ***tag_label*** names will be used in the post-processing module code to pass attribute information to the output data.

The 'file_alias' line identifies the Allied Model data file from which the data will be extracted, where the ***file_alias_name*** referes to a file identified in the 'files' section above. The 'field_info' line identifies the output FPA metafile. The ***element*** and ***level*** names in the 'field_info' declarations must be declared in the 'Elements' or 'Levels' blocks of a Configuration file! The user will have to include declarations in the Local Configuration file for any ***element*** or ***level*** data which is required by a new Allied Model but is not already declared. The user can refer to Config and Config.***name*** File Format, (Appendix A), as well as the examples given in the `Config.name` files in `$FPA/config`.

```
    metafiles      =
      {
      metafile_alias
        {
        #from files declaration, above
        file_alias           = file_alias_name
        #output data identified by element and level
        field_info           = element  level
        # optional: reset list of attributes
        attribute_info_reset = True | False
        # optional: output attributes from allied model
        # (may be specified more than once to generate a list)
        attribute_info       = tag_label  attrib_name  attrib_units
        # optional: reset list of default attributes
        default_attrib_info_reset = True | False
        # optional: default attributes set in output data
        # (may be specified more than once to generate a list)
        default_attrib_info       = attrib_name  attrib_value
        }
      } # End of metafiles Block
    } # End of allied_model Block
  } # End of NewModel Block
} # End of Sources Block
```

E.4 Changes to Allied Model Source Code

The code for the Allied Model itself is treated as much like a 'black box' as possible. The requirements are that the application is run as an independent program with all input and output to named files, and executed by a run string such as:

```
model_executable full_pathname_of_aliasfile
```

File information is passed to the application through the use of an aliasfile (usually created by the specialized pre-processing module), identified by a ***file_alias_name*** of 'aliases' in the 'files' declaration in the 'Sources' block of the Local Configuration file. The aliasfile is a table of file aliases and full pathnames for all input and output files used by the application. Most file aliases and pathnames are constructed directly from each ***file_alias_name*** section in the 'files' declaration.

Some code must be added to the Allied Model itself to access the aliasfile. Code for accessing Fortran routines is found in FTN_alias.f in the FPA Source Code. Routine FTN_MODEL_SETUP() is called at the start of the application to read the aliasfile for the aliases and pathnames. All files within the application are then identified by alias, with files being opened by calls to the routine FTN_MODEL_FILE_OPEN(), and all Logical Unit numbers for Fortran reads are also cross-referenced to the aliases.

Files that must be created for running the application can be produced through the pre-processing module or through an initialization module, which are both described below.

E.5 FPA Specialized Processing Modules

The following four sections describe the initialization (if required), pre-processing, execution, and post-processing modules required to integrate an Allied Model into the FPA processing environment. These modules are meant to be run as stand-alone programs. FPA routines common to most of these modules are described below.

- FPA Standard Library routines setup_files() and define_setup() are used to read the Local Setup file (local_setup) and to initialize access to the Local Configuration file.
- FPA Standard Library routine get_target_map() is used to set a map projection for the application data.
- FPA Standard Library routine get_source_info() is used to access the file and directory information for the Allied Model from the Local Setup and Local Configuration files.
- FPA Standard Library routine get_directory() is used to set the base Data Directory for the Allied Model.
- Some routines do not need to return all arguments; NULL is used to identify arguments which are not returned.

FPA Objects common to most of these modules are described below.Extensive use is made of two Objects in particular, the FLD_DESCRIPT Object and the PLOT Object (both described below).

- LOGICAL is a simple TRUE or FALSE as defined in types.h.
- STRING Objects are simple character strings as defined in types.h.

- POINT Objects are two element arrays for X, Y locations as defined in misc.h.
- FLD_DESCRIPT Objects contain all information required to identify individual fields in the FPA (including information about directory, field, and time) as defined in files_and_directories.h. Individual parameters in the FLD_DESCRIPT Object are set by calls to FPA Standard Library routine set_fld_descript(). Parameters may be blanked out by using the FpaCblank macro. Parameters which are not set are not changed.
- PLOT Objects contain arrays of data at single locations as defined in plot.h. Locations of sites for extracting data, input data for the application, and output data from the application are stored as
- PLOT Objects within the processing module code. The choice of PLOT Objects was to allow any combination of point data to be used, to produce code that could be easily modified and make use of FPA PLOT Object manipulation, and to simplify future display of the PLOT Object data inside the FPA. (Three general routines that are used are build_..._plot() to identify parameters to be stored in the PLOT Object, add_to_..._plot() to add parameters at a given location, and get_from_..._plot() to return parameters at a given location.)
- SURFACE Objects contain field data for B-spline surfaces as defined in surface.h.
- METAFILE Objects contain data for one or more fields as defined in metafile.h. METAFILE Objects contain information conforming to the Graphics Metafile Standard provided by the FPA Developers.

Routines exist to read three basic categories of files: FPA Configuration Type Files, ASCII Type Files, and Binary Type Files.

- Routines to read FPA Configuration Type files contain a loop with a call to FPA Standard Library routine getline() to read each line of the file. Blank lines or lines beginning with # are ignored. Lines in the file containing data are identified by a keyword which begins the line, and are read using FPA Standard Library routines string_arg(), int_arg(), long_arg(), float_arg(), or double_arg() depending on the type of word to be read. Each word is separated from the next by blanks or tabs. Each call to the FPA Standard Library routines reads one word and returns the remainder of the line; thus the line is read by successive calls. Since all data in the file is identified by keyword, the format of the file does not need to be known.
- Routines to read ASCII Type files contain successive calls to FPA Standard Library routine getline() for each line of the file to be read. Desired parameters are read using the same FPA Standard Library routines as with the FPA Configuration Type File (if the words are separated by white space), or using the FPA Standard Library routines fixed_string_arg(), fixed_int_arg(), fixed_long_arg(), fixed_float_arg(), or fixed_double_arg() (if the words are in a formatted file). However, the exact format of the file must be known, since all lines of the file must be read to extract the required data.
- Routines to read Binary Type files contain calls to system routine fread(), which will read blocks of data based on the given format. The format required by the application controls the parameters used.

Routines exist to write two basic categories of files; ASCII Type Files and Binary Type Files.

- Routines to write ASCII Type files contain calls to system routine fprintf(), based on the required format of each line of the file.
- Routines to write Binary Type Files use calls to system routine fwrite(), which will write blocks of data based on the given format.

E.5.1 Initialization for Allied Models

An initialization module should be used to create any constant files which are used by the application. This module is a stand-alone program, and is meant to be run only once, when the model is installed. It initializes input files for all subsources of the model. The program is run by:

```
allied_init <SETUP> <SOURCE>
```

E.5.2 Pre-processing for Allied Models

The pre-processing module is used to initialize the Data Directory and extract the input files for the application. This module is a stand-alone program, and is run each time the application is executed. The program will be run for each subarea by:

```
allied_prep <SETUP> <SOURCE> <SUBSOURCE> source_type <RTIME>
```

The FPA Standard Library routine prepare_source_directory() checks the 'Dstamp' files in the Data Directory structure to match the time at which the application is run (rtime). If the correct time is not found, the routine will automatically shuffle files down to a 'Prev' directory in the Data Directory structure to make room for the new data. (Note that an Allied Model cannot be run if the run time (rtime) is older than ALL the times in the Data Directory structure.)

E.5.3 Executing Allied Models

The execution module is used to run the actual Allied Model application. This module is a stand-alone program, and is run each time the application is executed. The program will be run for each subarea by:

```
allied_exec <SETUP> <SOURCE> <SUBSOURCE> <RTIME>
```

The time at which the application is run (rtime) is used to test the Data Directory to ensure that the pre-processing module has been run.

The execution module identifies the pathname to the Allied model program (identified by the program_alias 'Program' in the 'programs' section of the Local Configuration file), and the aliasfile required by the program (identified by the *file_alias_name* 'aliases'), and copies them to a buffer. The program is then started by a call to the FPA Standard Library routine shrun(), with the runstring:

```
"program_executable full_pathname_of_aliasfile"
```

E.5.4 Post-processing for Allied Models

The post-processing module is used to re-format Allied Model input or output files into files which can be used within the FPA. This module is a stand-alone program, and is run each time the application is executed. The program can be run for each subarea by:

```
allied_post <SETUP> <SOURCE> <SUBSOURCE> <RTIME>
```

The time at which the application is run (rtime) is used to find the appropriate Data Directory in which to write the FPA files. The data files to be produced are identified in the 'metafiles' section in the 'Sources' block of the Local Configuration file (described above).

The program loops through each time and each metafile, sets the identification field descriptor for the metafile, and extracts a METAFILE Object from the input or output files of the application. The routine, reads the files identified by the ***file_alias_name*** 'inputfile' or 'outputfile', extracts data for the required time as a PLOT Object, and copies the data into a METAFILE Object. The program then places a lock on the Data Directory using FPA Standard Library routine set_shuffle_lock(), builds a file name for the metafile using FPA Standard Library routine build_meta_filename(), and outputs the METAFILE Object to the file name using FPA Standard Library routine write_metafile().

Code also exists to extract SURFACE Object data from application files. The process is similar to producing a METAFILE containing a PLOT Object, except that the data for the required time must be converted to a SURFACE Object. This is done by converting the PLOT Object data to a grid, and identifying the location of each point on the FPA basemap with respect to this grid. For each point on the FPA basemap, a 4 by 4 subgrid of data around the point is extracted from the application data, the 4 by 4 subgrid data is converted into a SURFACE Object using FPA Standard Library routine grid_surface(), and the value at the point is evaluated using FPA Standard Library routine eval_surface(). When values at all points on the FPA basemap have been evaluated, the grid of values are then converted into a SURFACE Object using FPA Standard Library routine grid_surface(). Data at points within the area that the application covers are extracted from the application data, while data at points outside the area are extrapolated from values at the nearest edge of the application data (which gives the field a 'flattened' appearance beyond the data coverage).

E.6 Allowing User Access to Tested Allied Models

Once a new Allied Model has been developed and tested, the user may wish to make the Model available to be used by others. This can be done by signing on as root, becoming user bin by the command

```
su bin
```

and then making the following changes:

- Edit the file `$FPA/setup/Template` and add a line to the 'directories' block (as was done in the Local Setup File section above), as in:

```
amodels.test      your_directory/AModels.TEST
```

- Edit the file `$FPA/setup/Template` and add lines to the "[allied.model]" section of the 'interface' block (as was done in the Local Setup File section above), as in:

```
NewModel Area1
NewModel Area2
```

These changes will allow any future users of the FPA to "turn on" this Allied model when they run mkfpadb and customize their Local Setup file. The new Allied Model will then appear in the Products pull-down in the FPA.

Note
Any changes you make to the template setup file may be lost after upgrading FPA. Be sure to keep a copy for comparison after an update.

Current users of the FPA can also add the new Allied Model to their system by making the following changes while signed on to their users account:

- Edit their Local Setup file and add a line to the 'directories' block (as was done in the Local Setup File section above), as in:

```
amodels.test your_directory/AModels.TEST
```

- Edit their Local Setup file and add lines to the "[allied.model]" section of the 'interface' block (as was done in the Local Setup File section above), as in:

```
NewModel Area1
NewModel Area2
```

- Copy the 'Sources' block of the Local Configuration file (local_config) containing the new Allied Model to their Local Configuration file.
- Ensure that the user's path can access the actual Allied Model program and constant input data files in ***your_directory***/AModels.TEST, as well as the pre-processing, execution and post-processing modules in ***your_processing_directory***.

These changes will result in the new Allied Model appearing in the Products pull-down in the FPA for the current user.

E.7 FPAWarp (Fitting Point Data to a Field)

The FPA Warp model is an internal Allied Model designed to allow users to merge point data (whether at locations on a grid or random locations) with continuous field data. The FPA Warp model accomplishes this merge by calculating the differences between the point data values and the values from a "guess" continuous field at the same locations, and then applying adjustments iteratively until the continuous field values become reasonably close to the point data values. The user can access the FPA Warp model without the need to develop special code, by simply adding lines in the Local Setup file and the Local Configuration file.

- Edit the Local Setup file and add a line to the "[allied.model]" section of the 'interface' block as in:

```
NewFPAWarp
```

- Edit the Local Configuration file and add another 'Sources' block, as in:

```
Sources
  {
  NewFpaWarp
    {
    label            = "New FPA Warp"
    short_label      = "New FpaWarp"
    source_type      = Allied
    directory_tag    = AModels.Data
    directory_path   = FpaWarp
    directory_layers = 1
    subsources       = None
    allied_model     =
      {
      time_matching  = True
      source_info    = interp or depict
      process        = fpawarp_exec <SETUP> <SOURCE> <SUBSOURCE> <RTIME> \
                        tension influence file_creation \
                        ( guess_time  plot_time  output_time  ... )
      programs       = None
      files          = None
```

Note
The ***tension*** and ***influence*** parameters are used by the FPA Warp model to control the fit of the point data to the continuous field.

The ***tension*** parameter is used to control the accuracy of the fitting. The FPA Warp model calculates the mean absolute error between the point data values and the guess field values for each iteration. The iterations end when the mean absolute error no longer decreases, or when the change in the mean absolute error becomes less than the average error divided by the ***tension***. A larger value of ***tension*** will therefore result in more iterations, which should result in the point data being fitted more closely. (Note that the minimum allowed ***tension*** is 10.)

The ***influence*** parameter is used to control the effect of the point data on the guess field in data sparse areas. Each point data value will adjust the continuous field at nearby locations. However, if no data points are found to be closer to a given location than the number of FPA grid spaces given by ***influence***, then the continuous field will remain unchanged at that location. A smaller value of ***influence*** will therefore restrict adjustments to the continuous field to the immediate vicinity of the point data. (Note that the minimum allowed ***influence*** is 1 FPA grid space.)

The ***file_creation*** parameter controls whether an output file will be created if no acceptable plot data is found (YES or NO).

The times for which the FPA Warp model is run are normally the times available in the directory identified by the 'source_info' declaration. However, the "***..._time***" parameters give optional control over exactly which times are used for the guess field, the plot data fields, and the output metafiles. Each set of files required is identified by three times. The format for the times is given by one of the following:

- xx - for hours from the 'zero' hour in the Depiction sequence, that is, the time at which the application is run (RTIME)

- rr/hh - for the day (rr where 0 is today, 1 is tomorrow) and GMT hour of the day (hh)
- rr/hhL - for the day (rr where 0 is today, 1 is tomorrow) and local hour of the day (hh)

The 'required_fields' section below is used to identify the guess field for the continuous data (identified by the name 'guess_field'), and the file(s) containing the point data to merge with the guess field (identified by a name that begins with 'plot_field'). The location of these data files will default to the 'source_info' declaration given at the start of the 'allied_model' section unless it is reset here. The 'guess_field' and 'plot_field' information will be used in the FPA Warp model code to identify all required data. Note that the ***element***, ***level*** and ***source*** names in the 'field_info' declarations, must be declared in the 'Elements' or 'Levels' or 'Sources' blocks of a Configuration file! The user will have to include declarations in the Local Configuration file for any ***element***, ***level*** or ***source*** data which is required by a new FPA Warp model but is not already declared. The user can refer to Config and Config.***name*** File Format, (Appendix A), as well as the examples given in the `Config.name` files in `$FPA/config`.

```
required_fields    =
  {
  guess_field
    {
    # input data identified by element and level
    field_info     = element  level
    # optional: location of input field
    source_info    = source
    }
  plot_field
    {
    # input data identified by element and level
    field_info     = element  level
    # optional: member of a Plot Object
    sub_field_info = sub_field
    # optional: location of input field
    source_info    = source
    }
  } # End of required_fields Block
required_wind_crossrefs  = None
required_value_crossrefs = None
```

The 'metafiles' section below is used to identify the output FPA data produced by the new FPA Warp model. The location of the file will default to the 'directory_tag' given at the start of the 'Sources' declaration (that is, AModels.Data). The ***metafile_alias*** (which can be any name) will be used in the FPA Warp model code to identify the output data. Note that the ***element*** and ***level*** names in the 'field_info' declarations must be declared in the 'Elements' or 'Levels' blocks of a Configuration file! The user will have to include declarations in the Local Configuration file for any ***element*** or ***level*** data which is required by a new FPA Warp model but is not already declared. The user can refer to Config and Config.***name*** File Format, (Appendix A), as well as the examples given in the `Config.name` files in `$FPA/config`.

```
metafiles      =
  {
  metafile_alias
    {
```

```
          # output data identified by element and level
          field_info = element  level
          }
        } # End of metafiles Block
      } # End of allied_model Block
    } # End of NewFpaWarp Block
  } # End of Sources Block
```

E.8 FPA Create Area (Creating Discrete Areas From Objects)

The FPA Create Area model is an internal Allied Model designed to allow users to generate Discrete type fields from features in Line, Link Chain or Scattered type fields. The FPA Create Area model accomplishes this by using attributes from each source object to calculate the boundary of a discrete object. The user can access the FPA Create Area model without the need to develop special code, by simply adding lines in the Local Setup file and the Local Configuration file.

- Edit the Local Setup file and add a line to the "[allied.model]" section of the 'interface' block as in:

```
NewFPACreateArea
```

- Edit the Local Configuration file and add another 'Sources' block, as in:

```
Sources
  {
  NewFpaCreateArea
    {
    label             = "New FPA Create Area"
    short_label       = "New FpaArea"
    source_type       = Allied
    directory_tag     = AModels.Data
    directory_path    = FpaCreateArea
    directory_layers  = 1
    subsources        = None
    allied_model      =
      {
      time_matching   = True
      source_info     = interp or depict
      process         = fpacreate_area <SETUP> <SOURCE> <SUBSOURCE> \
                        <RTIME> <VTIME>  smoothing \
                          ( start_time  end_time  lchain_type  node_type )
      programs        = None
      files           = None
```

Note

The ***smoothing*** parameter is used by the FPA Create Area model to control the amount of detail in the discrete areas being created.

The ***smoothing*** parameter is used to control the detail in the discrete areas being created, with a larger values creating smoother discrete areas. The default ***smoothing*** value is 500, and the minimum allowed is 10.

The FPA Create Area model is run with data from the current depiction time, as given by the <VTIME> parameter.

The optional ***start_time*** and ***end_time*** parameters can be used with Link Chain type fields to set a subset of the track or nodes to use. The format for the times is given by hh or hh:mm where hh is hours and mm is minutes from the current depiction time, as given by the <VTIME> parameter.

The optional ***lchain_type*** parameter can be used with Link Chain type fields. A value of 'track' (the default) will create one area based on all the nodes of the link chain, while a value of 'nodes' will create one area for each node of the link chain.

The optional ***node_type*** parameter can be used with Link Chain type fields. A value of 'normal' (the default) will create areas based on the normal link chain nodes, while a value of 'interp' will create areas based on the interpolated link chain nodes.

The 'required_fields' section below is used to identify the field and attributes used to create the discrete areas. The location of these data files will default to the 'source_info' declaration given at the start of the 'allied_model' section unless it is reset here.

The ***required_field_alias*** (which can be any name) will be used in the FPA Create Area model code to identify all required data.

Note that the ***element***, ***level*** and ***source*** names in the 'field_info' declarations, must be declared in the 'Elements' or 'Levels' or 'Sources' blocks of a Configuration file! The user will have to include declarations in the Local Configuration file for any ***element***, ***level*** or ***source*** data which is required by a new FPA Create Area model but is not already declared. The user can refer to Config and Config.***name*** File Format, (Appendix A), as well as the examples given in the `Config.name` files in `$FPA/config`.

The ***field_type*** can be set to **Scattered** for field labels or left blank to use the field itself.

```
required_fields =
    {
    required_field_alias
      {
      field_info                = element  level [ field_type ]
      sub_field_info            = sub_field
      source_info               = source
      attribute_info            = tag_label  attrib_name  attrib_units
      ...
      node_attribute_info       = tag_label  attrib_name  attrib_units
      ...
      }
    } # End of required_fields Block
  required_wind_crossrefs  = None
  required_value_crossrefs = None
```

The 'attribute_info' and 'node_attribute_info' keywords are used to specify attribute information for the FPA Create Area model identified by specific ***tag_label*** names. You may also specify a ***tag_label*** name which can be referred to in the 'metafiles' section in order to pass attributes from the source field down to the output field.

Note
Unrecognized ***tag_label*** names in 'required_fields' that do not match ***tag_label*** names in 'metafiles' generate warnings.

Recognized ***tag_label*** names are listed below.

Note
The following listing shows recognized ***tag_label*** names for Scattered type fields, with areas created around each point location in the field

```
# If area is to be calculated based on distances from the
# point location in various directions (as noted below),
# then the tag_label names must be in cw or ccw order
  attribute_info          = point_n   attrib_name attrib_units
  attribute_info          = point_nne attrib_name attrib_units
  attribute_info          = point_ne  attrib_name attrib_units
  attribute_info          = point_ene attrib_name attrib_units
  attribute_info          = point_e   attrib_name attrib_units
  attribute_info          = point_ese attrib_name attrib_units
  attribute_info          = point_se  attrib_name attrib_units
  attribute_info          = point_sse attrib_name attrib_units
  attribute_info          = point_s   attrib_name attrib_units
  attribute_info          = point_ssw attrib_name attrib_units
  attribute_info          = point_sw  attrib_name attrib_units
  attribute_info          = point_wsw attrib_name attrib_units
  attribute_info          = point_w   attrib_name attrib_units
  attribute_info          = point_wnw attrib_name attrib_units
  attribute_info          = point_nw  attrib_name attrib_units
  attribute_info          = point_nnw attrib_name attrib_units
  attribute_info          = point_nnw attrib_name attrib_units

# If area is circular, the radius or diameter tag_label names can ↩
    be used
  attribute_info          = radius    attrib_name attrib_units
# or
  attribute_info          = diameter  attrib_name attrib_units
```

Note
The following listing shows recognized ***tag_label*** names for Line type fields, with areas created around each line in the field

```
# tag_label names are distances 90 degrees left or right
# wrt AttribLineDirection at each point
  attribute_info          = radius   attrib_name attrib_units
# or
```

```
          attribute_info          = diameter attrib_name attrib_units
        # or
          attribute_info          = left     attrib_name attrib_units
          attribute_info          = right    attrib_name attrib_units
```

Note
The following listing shows recognized ***tag_label*** names for Link Chain type fields, with areas created around each link chain in the field, or around each node of each link chain in the field

```
        # tag_label names are distances wrt an orientation (if given)
        # or 90 degrees left or right wrt AttribLnodeDirection
          node_attribute_info     = orientation attrib_name attrib_units
        # or
          node_attribute_info     = radius      attrib_name attrib_units
        # or
          node_attribute_info     = diameter    attrib_name attrib_units
        # or
          node_attribute_info     = left        attrib_name attrib_units
          node_attribute_info     = right       attrib_name attrib_units
```

The 'metafiles' section below is used to identify the output FPA data produced by the new FPA Create Area model. The location of the file will default to the 'directory_tag' given at the start of the 'Sources' declaration (that is, AModels.Data). The ***metafile_alias*** (which can be any name) will be used in the FPA Create Area model code to identify the output data. Note that the ***element*** and ***level*** names in the 'field_info' declarations must be declared in the 'Elements' or 'Levels' blocks of a Configuration file! The user will have to include declarations in the Local Configuration file for any ***element*** or ***level*** data which is required by a new FPA Create Area model but is not already declared. The user can refer to Config and Config.***name*** File Format, (Appendix A), as well as the examples given in the `Config.name` files in `$FPA/config`.

```
    metafiles      =
      {
      metafile_alias
        {
        # output field identified by element and level
        field_info          = element  level
        # list of attributes for output field
        attribute_info      = tag_label  attrib_name  attrib_units
        ...
        # list of default attributes for output field.
        default_attrib_info = attrib_name  attrib_value
        ...
        }
      } # End of metafiles Block
    } # End of allied_model Block
  } # End of NewFPACreateArea Block
} # End of Sources Block
```

Each 'attribute_info' line specifies an attribute to include in each new discrete area. The ***tag_label*** names listed here must also be listed in the 'required_fields' section.

Each 'default_attrib_info' line specifies an ***attrib_name*** and a default ***attrib_value*** to include in each new discrete area.

E.9 FPA Create Contour (Creating Discrete Areas From Contours)

The FPA Create Contour model is an internal Allied Model designed to allow users to generate Discrete type fields from the contours of Continuous type fields. The FPA Create Contour model accomplishes this by using minimum and maximum contour values to create the boundary of a discrete object. The user can access the FPA Create Contour model without the need to develop special code, by simply adding lines in the Local Setup file and the Local Configuration file.

- Edit the Local Setup file and add a line to the "[allied.model]" section of the 'interface' block as in:

```
NewFPACreateContour
```

- Edit the Local Configuration file and add another 'Sources' block, as in:

```
Sources
  {
  NewFpaCreateContour
    {
    label             = "New FPA Create Contour"
    short_label       = "New FpaCont"
    source_type       = Allied
    directory_tag     = AModels.Data
    directory_path    = FpaCreateCont
    directory_layers  = 1
    subsources        = None
    allied_model      =
      {
      time_matching   = True
      source_info     = interp or depict
      process         = fpacreate_cont <SETUP> <SOURCE> <SUBSOURCE> <RTIME>
      process         = fpacreate_cont <SETUP> <SOURCE> <SUBSOURCE> \
                        <RTIME> smoothing  min_points \
                          min_value  max_value  units ( valid_times )
      programs        = None
      files           = None
```

Note
The ***smoothing*** and ***min_points*** parameters are used by the FPA Create Contour model to control the amount of detail in the discrete areas being created.

The ***smoothing*** parameter is used to control the detail in the discrete areas being created, with a larger values creating smoother discrete areas. The default ***smoothing*** value is 500, and the minimum allowed is 10.

The ***min_points*** parameter sets the minimum number of points allowed, to prevent very small discrete areas (and holes) being created. The default ***min_points*** value is 10.

The ***min_value***, ***max_value***, and ***units*** parameters set the range of contour values for creating the discrete areas. A value of '-' for the ***min_value*** parameter will create discrete areas within regions below the ***max_value***. A value of '-' for the ***max_value*** parameter will create discrete areas within regions above the ***min_value***.

The times for which the FPA Create Contour model is run are normally the times available in the directory identified by the 'source_info' declaration. However, the ***valid_times*** parameter gives optional control for specifying one or more times to be used for extracting discrete areas. The format for the times is given by one of the following:

- xx - for hours from the 'zero' hour in the Depiction sequence, that is, the time at which the application is run (RTIME)
- rr/hh - for the day (rr where 0 is today, 1 is tomorrow) and GMT hour of the day (hh)
- rr/hhL - for the day (rr where 0 is today, 1 is tomorrow) and local hour of the day (hh)

The 'required_fields' section below is used to identify the continuous field containing the contours to be extracted. The location of these data files will default to the 'source_info' declaration given at the start of the 'allied_model' section unless it is reset here.

The ***required_field_alias*** (which can be any name) will be used in the FPA Create Contour model code to identify all required data.

Note that the ***element***, ***level*** and ***source*** names in the 'field_info' declarations, must be declared in the 'Elements' or 'Levels' or 'Sources' blocks of a Configuration file! The user will have to include declarations in the Local Configuration file for any ***element***, ***level*** or ***source*** data which is required by a new FPA Create Contour model but is not already declared. The user can refer to Config and Config.***name*** File Format, (Appendix A), as well as the examples given in the `Config.name` files in `$FPA/config`.

The ***field_type*** can be set to **Scattered** for field labels or left blank to use the field itself.

```
required_fields =
    {
    required_field_alias
      {
      field_info              = element  level [ field_type ]
      sub_field_info          = sub_field
      source_info             = source
      }
    } # End of required_fields Block
  required_wind_crossrefs  = None
  required_value_crossrefs = None
```

The 'metafiles' section below is used to identify the output FPA data produced by the new FPA Create Contour model. The location of the file will default to the 'directory_tag' given at the start of the 'Sources'

declaration (that is, AModels.Data). The ***metafile_alias*** (which can be any name) will be used in the FPA Create Contour model code to identify the output data. Note that the ***element*** and ***level*** names in the 'field_info' declarations must be declared in the 'Elements' or 'Levels' blocks of a Configuration file! The user will have to include declarations in the Local Configuration file for any ***element*** or ***level*** data which is required by a new FPA Create Contour model but is not already declared. The user can refer to Config and Config.***name*** File Format, (Appendix A), as well as the examples given in the `Config.name` files in `$FPA/config`.

```
    metafiles      =
      {
      metafile_alias
        {
        # output field identified by element and level
        field_info          = element  level
        # list of default attributes for output field
        default_attrib_info = tag attrib_name attrib_units
        ...
        }
      } # End of metafiles Block
    } # End of allied_model Block
  } # End of NewFPACreateArea Block
} # End of Sources Block
```

Each 'default_attrib_info' line specifies an ***attrib_name*** and a default ***attrib_value*** to include in each new discrete area.

Appendix F

Predefined GUI Selection Lists

Predefined GUI selection lists are lists which appear in the graphical user interface but are preset by the system manager and cannot be modified by the user from within FPA. The selection of a particular list is controlled by the setup file for the particular instance of FPA.

Note
If a label in any of the following files is only required to be in one language then the label is taken as presented. If multi-language labels are required then the label line must be specified as:

```
<*default*> "a b c" <*language*> "a ä u" <*language*> "...
```

Where: <*default*> specifies the label to use if none of the other specifiers are valid. <*language*> is the required language as specified in the `LANG` environment variable.

Selection lists are located in `$FPA/setup/preset_lists/TEMPLATE` and should be copied to a user's local setup directory by typing:

```
cp $FPA/setup/preset_lists/TEMPLATE/* $HOME/setup/preset_lists/
```

You may then customize the list for that user according to the formats described below.

Warning
It is common practice in technical writing to encase optional arguments within square brackets ([]). In the following list formats square brackets are used to encase a list key. The reader may find this confusing. Assume the traditional meaning unless the text indicates otherwise.

F.1 Area Sample Filter Lists

The file must be named `area_sample_filters` When sampling areas the list of attributes associated with a particular area can be quite large. The purpose of this file is to:

- provide a mechanism to filter the attribute display to show only those attributes of interest.

- set the display order of the attributes to something other than that in the configuration files.

The order that the attributes will appear in the list is that of the filter list and not that of the attribute order in the configuration file. The lists are separated by list keys (element-level pairs) in square brackets as:

```
[element.level] filter_label
    attribute
    attribute
    ...
[element.level] filter_label
    attribute
    ...
```

The element-level pairs specify that the filter is for a specific field. The element and level names are those defined in the configuration files. If the filter is to be used in every field, then the element-level pair identifier is replaced with the keyword "ALL_FIELDS". Don't forget the period between the element and level!

Example F.1 Resetting "ALL_FIELDS"

The filter label is whatever you want it to be. So to always show a list of certain attributes the file might, for example, contain:

```
[ALL_FIELDS] "Main Values"
    cloud_base_1
    cloud_base_2
    visibility
```

Example F.2 Sample Filter

To associate a particular filter list with just one field we would have something like the following:

```
[weather_synoptic.surface] "Weather Values"
another list
```

There is a special filter group which is always included in the display called "All Attributes" which will display all of the attributes. You may want to still limit which attributes are displayed with this filter and so you can override this by including the group "ALL_ATTRIBUTES" in this file. A label is not required for this group as the default name is used, but if you want to change the label you may.

This list is different in that it contains all attributes which are NOT to be displayed. For Example:

```
[ALL_ATTRIBUTES] "Show Everything"
    attributes not to display
```

Will change the label from "All Attributes" to "Show Everything" and remove from display the given list of attributes.

F.2 Depiction Field Visibility

The file must be named `field_visibility`. This file contains the definitions of predefined lists of depiction field visibility states. The lists are separated by list keys in square brackets as:

```
[list_1_key] list_label
    field_group_id always on
    element level always on
    element level always on
[list_2_key] list_label
    element level
```

and so on. The list keys should be something that makes sense, as **xfpa**, if run with the `-v` option will come up with that visibility list. i.e. `xfpa -v` ***list_2_key*** would come up with the second list selected already.

The ***list_label*** is what the user will see as the selection in the pulldown list of predefined visibility states.

The remaining lines are either the identifiers, as recognized in the configuration files, of the field groups or the fields themselves. Any field not listed in the group will be set to off. The default state is for the field to be visible when the associated group is visible. If the field is to be visible always no matter what, then the element and level are followed by the optional key words "always on".

Example F.3 Depiction field visibility

```
[marine] Marine Fields
    pressure msl always on
    weather sfc
    wind sfc
```

F.3 Guidance Field List

This file must be named `guidance`. This file contains the definitions of predefined lists of guidance fields as seen under the **Guidance** → **Select** pulldown menu. Any lists incorporated into the guidance selection via this mechanism are fixed and cannot be deleted or have fields added to or removed from the list. This list can, however, be copied and the copy modified. The list files to incorporate into the guidance selection are specified in the setup file.

The lists are separated by list keys in square brackets as:

```
[list_1_key] list_label
    field
    field
[list_2_key] list_label
    field
```

and so on. The contents of the list key separator can be anything that make sense as they are not used at the moment.

The list label is what the user will see to identify this particular list. The field lines consist of:

```
element level source sub-source previous
```

Where ***element***, ***level***, ***source*** and ***sub-source*** are as specified in the configuration files. Previous, which only has meaning when the source is depict or interp, indicates that the depiction previous to the currently selected one is required. If previous is specified then sub-source must be set to be dash "-". Note that not all sources have sub-sources.

Example F.4 Guidance field list

```
[list1] 500Mb Fields
    height        500 FEM
    height_change 500 FEM
    vertical_vel  500 FEM
```

F.4 Point Lists

This file must be named `point_lists`. This file contains the definitions of predefined lists of points which will be used under both the sample and label edit functions. There is only one instance of this file permitted. The lists are separated by list keys in square brackets as:

```
[list_1_key] list_label
    latitude longitude comments
    latitude longitude comments
[list_2_key] list_label
    latitude longitude comments
```

and so on. The contents of the list key separator can be anything that make sense.

The list_label is what the user will see to identify the list.

The format of the latitude and longitude must be given as one of

- [+|-]DDD[N|E|W|S] = whole degrees
- [+|-]DDD.ddd[N|E|W|S] = decimal degrees
- [+|-]DDD:MM[:SS][N|E|W|S] = degrees, minutes, seconds
- [+|-]DDDMM['SS["]][N|E|W|S] = degrees, minutes, seconds

If the direction is given as one of N|E|W|S then the leading sign is not required. Remember west and south are negative!

A comment may follow the position if it is proceeded by a pound sign "#". This is for file maintenance only and has no other purpose.

Example F.5 Point lists

```
[main_cities] Main Cities
    43:70N 79:60W    # Toronto
    42:35N 83:00W    # Winsor
    46:40N 84:50W    # Sault St Marie
```

F.5 Field Update Office Default

This file must be named `field_update_office_default`. Only one instance of this file is permitted. It lists the fields and times that an office loads by default from a particular source. This does not mean that these depictions will be updated automatically, it simply sets the default selection. The user still has the opportunity to fine tune the default selection. The lists are separated by source keys in square brackets as:

```
[source]
    element level (list of valid times relative to run time of the source)
  ...
[source]
    element level (list of valid times relative to run time of the source)
  ...
```

and so on. The sources must be defined in a configuration file, as do the element and level identifiers.

Example F.6 Field update office default

```
[GEM_REG]
    temperature  surface 00 06 12 18 24 36 48
    rel_humidity surface 18 42
[GEM_GLB]
    pressure     msl     00 06 12 18 24 30 36 42 48 54 60 66 72
    temperature  surface 54 60 66 72
    rel_humidity surface 66
```

Appendix G

Setup File Directory Override Options

G.1 Directories Block

The directories block of the setup file defines the locations of the data directories used by FPA. Each entry consists of a keyword recognized by the system to identify the directory, followed by the directory path. The path may contain UNIX environment variables, the most common of which are `$HOME` and `$FPA`.

Example G.1 Location of FPA database associated with this setup file

```
directories
{
  home    $HOME/MARITIMES
}
```

Normally the directories block of the setup file will contain only one entry. This being the home directory to which all relative paths will be interpreted as being relative to. The library software which reads this file does a `chdir()` to the home directory automatically.

If home is omitted from this block, no `chdir()` will be done. Instead, all relative paths will be interpreted as being relative to the directory defined by the `$FPA` environment variable (or `$HOME` if `$FPA` is not defined), and will be converted to absolute paths by concatenating this directory.

There are many other directories which can be specified but which are normally defaulted. The following list provides the default directory information. If any one of these is to be set to something other than the default then an entry is made in the directories block after the home directory entry.

The first column in the list is the key used by the system software to identify the directory. The second column provides a description of what the directory is used for and the default path(s) of the directory is given as a bulleted item.

Including a directory keyword and its associated directory can have one of two consequences:

- The setup file entry replaces the default directory path. The keywords associated with this action are shown in Table G.1.

- The setup file entry becomes the first member of a list of directories which are searched in order. The first instance of a required file found is used. This allows local versions or temporary test versions of some files to be used by the system while allowing others to default. Note that there is more than one default directory which are searched in the order given in the bulleted list. The keywords associated with this action are shown in Table G.2.

Table G.1: Directories Replaced by Setup File Entry

Keyword	**Directory description and defaults**
AModels.Data	Allied Model data • `AModels.DATA`
AModels.Exec	Allied Model executables • `$FPA/AModel.EXEC`
config	Base directory for all configuration files. • `$HOME/config`
ctables	Base directory for all colour tables for imagery config files. • `$HOME/config/ctables`
Data	Base directory for all datafiles. • `Data`
ExternalDepictions	Base directory for all external depiction databases. • `EXTERNAL`
Guidance	Base directory for NWP guidance. • `Guidance`
help.source	Location of all of the online help files. • `$FPA/doc/online`
images	Base directory for all image files. • `$HOME/images`
ingest.src	Location of GRIB data files required by the ingest process. • `$FPA_LOCAL_GRIB`

Table G.1: (continued)

Keyword	Directory description and defaults
ingest.stat	Location of the ingest status file. • `Guidance`
ingest.log	Location of the ingest log file. • `Guidance`
Maps	Location for map background files and overlays • `Maps` • `$FPA/data/common/CommonMaps`
psmet	PostScript graphical product generator (PSMet) root directory • `$HOME/setup/pdf/psmet`
psout	Location of output files created by PSMet • `PSOut`
setup	Location of FPA database setup files • `$HOME/setup`
svgmet	SVG graphical product generator (SVGMet) root directory • `$HOME/setup/pdf/svgmet`
svgout	Location of output files created by SVGMet • `SVGOut`
texmet	Root directory of the TexMet program. • `$HOME/setup/pdf/texmet`
texout	Location of TexMet output files. • `TexOut`

Table G.2: Setup File Entry Searched in Addition to Default Directory

Keyword	Directory Description and Default
memory.cfg	Location of area editor preset memory files. • `$HOME/config/Memory` • `$FPA/config/Memory`
menus.cfg	Location of entry menu configuration files • `$HOME/config/Menus` • `$FPA/config/Menus`
metafiles	Location of product definition files for the transmission of metafiles as a graphical product. • `$HOME/setup/pdf/metafiles`
patterns	Location of pattern files. • `$HOME/config/patterns` • `$FPA/config/patterns`
preset_lists	Location of files which define preset lists of information as used in the Graphical User Interface. • `$HOME/setup/preset_lists` • `$FPA/setup/preset_lists`
psmet_symbols	Location of symbol files used by PSMet • `$HOME/setup/pdf/psmet/common/ps` • `$FPA/setup/pdf/psmet/common/ps`
svgmet_symbols	Location of symbol files used by SVGMet • `$HOME/setup/pdf/svgmet/common/svg` • `$FPA/setup/pdf/svgmet/common/svg`

G.2 Configuration Files Block

This block specifies the names of the configuration files used by various processes. Relative paths are taken with respect to the directory defined by the config keyword above while absolute paths are just that.

Table G.3: Configuration file names

Keyword	File description and default name
config	File used to configure FPA • `Config`
gribs	Configuration used by the **gribin** ingest program (GRIB edition 0&1) • `$FPA/config/Gribs`
image	File used to configure imagery for FPA • `Image`
ingest	Configuration used by **gribin2** ingest program (GRIB editions 0,1&2) • `Ingest`
presentation	The presentation specification file • `Presentation`

Appendix H

User Defined Functions

The directory `$FPA/templates/userlib` contains template files (ANSI-C source, scripts and make-files) for building your own local client-written library, that can be accessed by FPA applications.

Do not modify the original files found in `$FPA/templates`. These files will be updated with future releases and patches. It is intended that you copy them to a convenient directory, and build the library there. The script copyuserlib has been provided for this purpose.

This directory should contain the following files:

File name	Description
`README`	File describing the contents of the directory
copyuserlib	script to copy the template files to a working directory
`user_rules.c`	source template for rules to calculate attribute values
`user_confirm.c`	to confirm that the correct user defined library is being accessed
`user_values.c`	source template for value calculations
`user_winds.c`	source template for wind calculations
libmake	script to build library
libput	script to place library where it can be accessed
librestore	script to restore the previous version of the library for use by the operational FPA software
`Makefile.lib`	make file used by libmake

Once you have copied the template files to a convenient working directory, instructions for writing your own functions are given inside the individual ".c" files.

Version Confirmation

The source file `user_confirm.c` is designed to confirm that the correct user defined library is connected. With each new major release of your library you should set an identifying message here. When the library is invoked this message will appear amongst the other startup messages in the log. This allows you to confirm functions in your new library are indeed being invoked. You can also set flags for displaying rules, value functions, and wind functions in the log. A flag set to `FALSE` implies that module does not need to be displayed in the startup message.

User defined Rules

The source file `user_rules.c` is designed to access user defined routines to determine attributes for meteorological fields. To add a new rule, you must follow the steps documented in the file: Add a prototype for the rule, add the rule to the rule table and finally add the code to calculate the rule. The template file contains three sample rules: clds_and_wx_rule, full_weather_rule, and full_cloud_rule.

User defined value functions

The source file `user_values.c` was designed to access user defined routines to extract values from fields of meteorological data. There is one example routine for value extraction called `values_for_values()`. It gives an example of how data can be calculated by passing values to a function that the user would supply.

User defined wind functions

The source file `user_winds.c` was designed to access user defined routines to extract wind speed and direction from fields of meteorological data. There are three example routines given for winds extraction: `winds_from_equations()` gives an example of winds calculated using a generic equation from the configuration files; `winds_from_values()` gives an example of winds calculated by passing values to a function that the user would supply; `gradient_wind_function()` gives an alternate way to calculate a gradient wind, using a more complicated function call rather than the equations given in the configuration files. This last function also allows a cross isobaric estimate to be added.

H.1 Testing and Installing the user library

To test the library before installing it as the operational version, build the library using **libmake**, then set and export the `FPA_SHLIB_PATH` variable to the directory in which the library was built. Running any FPA application from that environment will access the newly built library.

Once tested, you may then install the library in the standard directory, using the **libput** script. The **libput** script will fail if the shared library is in use, so be sure to shutdown the depiction editor and ingest daemon before you transfer the files. Unset the `FPA_SHLIB_PATH` variable and test that the library was transferred properly. The **librestore** script provides the ability to go back to the last working version of the library, if it turns out to have been installed prematurely!

H.2 Writing user defined rules in python

Python is an easy to learn programming language. It comes standard on most Linux distributions. It is interpreted so it need not be compiled and thus allows for easy testing and debugging. These features make it ideal for writing FPA rules. Rules are simple scripts that determine the value of field attributes based on the values of other field attributes. With the introduction of python rules, it should be easier for FPA users to harness the power of attribute rules for their databases. There are some sample rules located in `$FPA/bin`. You should not edit these as they may be overwritten the next time you update FPA. Instead copy them to your local bin directory `$HOME/bin` and edit them there. Once you have fully tested and debugged them you may wish to save them in the `$FPA/localbin` directory. This directory appears before `$FPA/bin` in the `$PATH` variable and so will be found before any of the default scripts. Common variables and tests used in User Defined rules have been included in the FPAlib.py file. To invoke python

rules use the **python_type_rules** or **python_entry_rules** in the Configuration file instead of **type_rules** or **entry_rules**.

Appendix I

Index