

**MASTER ENVIRONMENTAL LIBRARY
(MEL)**

**USER MANUAL
FOR THE
GRIDDED BINARY (GRIB)
SOFTWARE**



OCTOBER 15, 1998

**DEFENSE MODELING AND SIMULATION OFFICE
ALEXANDRIA, VA**

MASTER ENVIRONMENTAL LIBRARY
(MEL)

**USER MANUAL
FOR THE
GRIDDED BINARY (GRIB)
SOFTWARE**

VERSION 1.0

OCTOBER 15, 1998

MAR, INC.
99 PACIFIC ST., SUITE 200E
MONTEREY, CA 93940

PREPARED BY:

SAIC
550 CAMINO EL ESTERO, SUITE 205
Monterey, CA 93940

This page intentionally left blank

GRIB USER MANUAL

FOREWORD

The Department of Defense (DoD) Modeling and Simulation Master Plan (MSMP), DoD 5000.59-P, October 1995, assigned execution responsibility to Executive Agents and established sub-objectives for providing authoritative representations of the four natural environment domains (terrain, oceans, atmosphere, and space). Through separate letters, the Under Secretary of Defense for Acquisition and Technology (USD(A&T)) designated the Defense Mapping Agency (now part of the National Imagery and Mapping Agency), Department of the Air Force, and Department of the Navy as Modeling and Simulation Executive Agents (MSEAs) for terrain, atmosphere and space, and oceans, respectively.

The Master Environmental Library (MEL) project is part of the Modeling and Simulation (M&S) community's initiative to provide an authoritative representation of the natural environment. The MEL Gridded Binary (GRIB) software is one of the supplementary tools that complement the MEL project.

This document will be reviewed and updated by the MSEAs as required to maintain its currency. Comments and recommendations should be forwarded for review and possible inclusion to:

Office of the Ocean Executive Agent
1800 N. Beauregard St.
Suite 300
Alexandria, VA 22311
(703) 575-2880

This page intentionally left blank

GRIB USER MANUAL

RECORD OF CHANGES

Change Number	Date of Change	Change Description	Date Entered	Entered by

This page intentionally left blank

GRIB USER MANUAL

TABLE OF CONTENTS

FOREWORD	iii
RECORD OF CHANGES	v
TABLE OF CONTENTS	vii
SECTION 1. SCOPE	1
1.1 IDENTIFICATION.....	1
1.2 SYSTEM OVERVIEW.....	1
1.3 DOCUMENT OVERVIEW.....	1
SECTION 2. REFERENCED DOCUMENTS	3
2.1 GOVERNMENT DOCUMENTS.....	3
2.1.1 <i>Standards</i>	3
2.1.2 <i>Other Documents</i>	3
2.2 NON-GOVERNMENT DOCUMENTS.....	3
SECTION 3. SOFTWARE SUMMARY	5
3.1 SOFTWARE APPLICATION.....	5
3.2 SOFTWARE INVENTORY.....	5
3.3 SOFTWARE ENVIRONMENT.....	5
3.4 SOFTWARE ORGANIZATION AND OVERVIEW OF OPERATION.....	5
3.4.1 <i>Encoding A GRIB Message</i>	6
3.4.2 <i>Decoding A GRIB Message</i>	8
3.5 CONTINGENCIES/ALTERNATE STATES AND MODES OF OPERATION.....	9
3.6 SECURITY AND PRIVACY.....	9
3.6.1 <i>US Government System</i>	9
3.6.2 <i>Limits Of Liability</i>	9
3.6.3 <i>Limits Of Endorsement</i>	10
3.7 PROBLEM REPORTING AND RECOMMENDATIONS.....	10
SECTION 4. ACCESS TO THE SOFTWARE	11
4.1 FIRST-TIME USER OF THE SOFTWARE.....	11
4.1.1 <i>Equipment Familiarization</i>	11
4.1.2 <i>Access Control</i>	11
4.1.3 <i>Installation And Setup Of The MEL GRIB Software Library</i>	11
4.1.3.1 <i>Copy the Source Code</i>	11
4.1.3.2 <i>Setting the GRIB Environment</i>	12
4.1.3.3 <i>Building the Library</i>	13
4.2 INITIATING A SESSION.....	14
4.3 STOPPING AND SUSPENDING WORK.....	14
SECTION 5. PROCESSING REFERENCE GUIDE	15
5.1 CAPABILITIES.....	15
5.1.1 <i>Features</i>	15
5.1.1.1 <i>Library API</i>	15
5.1.1.2 <i>Additional Header Information</i>	15
5.1.1.3 <i>External Tables</i>	16
5.1.2 <i>Limitations</i>	16

GRIB USER MANUAL

5.1.2.1	Grid Definition Section Limitation	16
5.1.2.2	Packing Methods	16
5.2	CONVENTIONS	16
5.3	LEARNING GRIB USING GRIBSIMP	17
5.3.1	Overview Of GRIBSIMP	17
5.3.2	Command Line Options	18
5.3.3	Examples Using GRIBSIMP	22
5.3.3.1	Example 1: Decoding your first message	22
5.3.3.2	Example 2: Adding Indexing and Additional Printing	22
5.3.3.3	Example 3: Controlling the decoding process	23
5.3.3.4	Example 4: Creating a GrADS data set	24
5.3.3.5	Example 5: A Second GrADS Example	27
5.3.3.6	Example 6: Creating a Vis5D data set	30
5.4	DECODING EXAMPLE	30
5.4.1	Program 'decoder_ex.c'	31
5.4.2	Program 'getgribiee.c'	37
5.5	GRIB ENCODING	40
5.5.1	Introduction	40
5.5.2	Describing Data	41
5.5.3	Encoding Examples	43
5.5.4	Encoding Code Examples	43
5.5.4.1	Example 1 - encoder_ex1	43
5.5.4.2	Example 2 - encoder_ex2	47
5.5.4.3	Example 3 - encoder_ex3	50
5.6	GRIB TABLE MANAGEMENT	58
5.6.1	External Table Format	59
5.6.1.1	The Decoder Table File	60
5.6.1.2	The Encoder Table File	64
5.7	FUNCTION DEFINITIONS	67
5.7.1	Decoding Functions	67
5.7.1.1	init_gribhdr	67
5.7.1.2	free_gribhdr	67
5.7.1.3	init_dec_struct	68
5.7.1.4	init_enc_struct	68
5.7.2	Decoding	68
5.7.2.1	grib_dec	68
5.7.2.2	ld_dec_lookup	69
5.7.2.3	grib_seek	70
5.7.3	Encoding	71
5.7.3.1	grib_enc	71
5.7.3.2	ld_enc_config	72
5.7.3.3	ld_enc_ieeef	73
5.7.3.4	ld_enc_ffinfo	73
5.7.3.5	ld_enc_geomfile	74
5.7.3.6	map_parm	74
5.7.3.7	map_lvl	75
5.7.3.8	ld_enc_lookup	76
5.7.4	User Convenience Functions	76
5.7.4.1	gribhdr2file	76
5.7.4.2	apply_bitmap	77
5.7.4.3	display_gribhdr	78
5.7.4.4	hdr_print	78
5.7.4.5	make_default_grbfn	79
5.7.4.6	make_grib_log	79
5.7.4.7	prt_inp_struct	80
5.7.4.8	init_struct	81
5.7.5	Structures	81

GRIB USER MANUAL

5.7.5.1 The USER_INPUT structure (from input.h)	81
5.7.5.2 The DATA_INPUT structure (input.h).....	83
5.7.5.3 The GEOM_IN structure (input.h)	85
5.7.5.4 The GRIB_HDR structure (grib.h)	88
5.8 RELATED PROCESSING.....	89
5.9 DATA BACKUP.....	89
5.10 RECOVERY FROM ERRORS, MALFUNCTIONS, & EMERGENCIES.....	89
5.11 MESSAGES.....	89
APPENDIX A. GRIB MESSAGE STANDARD	91
APPENDIX B. MEL GRIB SOFTWARE LIBRARY INVENTORY	93
APPENDIX C. GRIB EXTENSIONS	97
APPENDIX D. RUNNING GRIB EXAMPLES.....	99
D.1 RUNNING 'DECODER_EX'	99
D.2 RUNNING 'GETGRIBIEEE'	99
D.3 RUNNING 'ENCODER_EX1'	100
D.4 RUNNING 'ENCODER_EX2'	100
D.5 RUNNING 'ENCODER_EX3'	101
APPENDIX E. GRIB SOFTWARE CONVENTIONS.....	103
E.1 DEFAULT FILENAME FOR ENCODED GRIB MESSAGES.....	103
E.2 THE IEEE FILES	103
E.3 DECODED IEEE FILE NAMES (FROM GRIBSIMP).....	104
E.4 DEFAULT DECODER TABLE FILE NAME CONVENTIONS	105
APPENDIX F. MEL GRIB SOFTWARE LIBRARY ERROR MESSAGES	107
APPENDIX G. ACRONYMS/ABBREVIATIONS.....	111
INDEX	113

LIST OF FIGURES

FIGURE 1. GRIB ENCODING PROCESS	7
FIGURE 2. GRIB DECODING PROCESS	8

LIST OF TABLES

TABLE 1 . CHOOSING SYSTEM CONFIGURATION DURING INSTALLATION.....	13
TABLE 2. SUB-TABLE IDENTIFIERS.....	64

GRIB USER MANUAL

NOTE: Conventions

This manual uses the following typographical conventions:

CAPITAL LETTERS for the names of Internet protocols, acronyms, and abbreviations (see **Appendix G**).

Boldface type for headings and references to other sections in this manual.

Italics for emphasis and publication titles.

Monospaced font for keywords in computer system commands, directory path names, and file names. In proper context, the text in [square brackets] represents command options and text in <angle brackets> represents items the user should replace with applicable text.

Monospaced italic font for Internet addresses.

SECTION 1. SCOPE

1.1 IDENTIFICATION

This document pertains to the Master Environmental Library (MEL) Gridded Binary (GRIB) Software version 3.0. This version of the GRIB data exchange format encoder/decoder software is recommended for use in conjunction with the MEL.

1.2 SYSTEM OVERVIEW

GRIB is a standard format designed by the World Meteorological Organization (WMO) to support the efficient transmission and storage of gridded meteorological data. Formally known as a general purpose, bit-oriented data exchange format, GRIB was approved by the WMO in 1985 and designated *FM 92-VIII Ext. GRIB (GRIdded Binary)*. Changes and extensions approved in 1990 were incorporated into *GRIB, Edition 1*.

A GRIB message consists of a series of header sections, followed by a bitstream of packed data representing one two-dimensional (2-D) grid of data values. The header sections are intended to fully describe the data included in the bitstream, specifying information such as the parameter, units, and precision of the data, the grid system and level type on which the data is provided, the process used to generate the data, and the date and time for which the data are valid. Additional descriptors may be added through the definition of extensions, but any changes to the standard must be backward compatible, so that a GRIB message can be read by any standard GRIB decoder.

Non-numeric descriptors are enumerated in tables, such that a 1-byte code in a header section refers to a unique description. The WMO provides a standard set of enumerated parameter names and level types, but the standard also allows for the definition of locally used parameters, geometries, and models. Any activity that generates and distributes GRIB messages must also make their locally defined GRIB tables available to users.

1.3 DOCUMENT OVERVIEW

This manual applies to the MEL GRIB Software Library 3.0, which provides a set of C functions for encoding and decoding GRIB messages. **Section 3** provides an overview of the features and limitations of this software implementation of the WMO GRIB code. Instructions for installing and testing the library are included in **Section 4**. The standard MEL GRIB decoder, `gribsimp`, is presented in **Section 5** to introduce the GRIB message structure and typical procedures in handling GRIB messages. **Section 5** provides the details of the programming interface to the MEL GRIB library, including a series of sample encoding and decoding programs listed in **Appendix D**. The complete structure of a standard GRIB message is presented in **Appendix A**, while **Appendix B** lists the MEL

GRIB USER MANUAL

GRIB Software Library inventory and **Appendix C** covers GRIB extensions. GRIB software conventions are detailed in **Appendix E** and possible error messages in **Appendix F**.

For a complete description of the WMO GRIB standard, including all standard parameter and level definitions, see *WMO Manual 306: Manual on Codes, Volume I.2*.

SECTION 2. REFERENCED DOCUMENTS

2.1 GOVERNMENT DOCUMENTS

2.1.1 STANDARDS

None

2.1.2 OTHER DOCUMENTS

- a. Department of Defense. Modeling and Simulation Master Plan, DoD 5000.59-P. Washington, DC: DoD, October 1995.
- b. Defense Modeling and Simulation Office (DMSO). MEL Software User Manual. Arlington: DMSO, 10 July 1997.

2.2 NON-GOVERNMENT DOCUMENTS

- a. World Meteorological Organization. WMO Manual 306: Manual on Codes. 2 vols. Geneva: WMO, 1988/1991/1997.
- b. WMO. World Weather Watch Technical Report No. 17: Guide to WMO Binary Code Forms. 2 parts. Geneva: WMO, May 1994.
- c. Doty, Brian. The Grid Analysis and Display System (1995) : n. pag. Online. Internet. 27 May 1998. Available:
ftp://grads.iges.org/grads/sprite/doc/gadoc151.txt

This page intentionally left blank

SECTION 3. SOFTWARE SUMMARY

3.1 SOFTWARE APPLICATION

The MEL GRIB Software Library software is used to encode and decode gridded meteorological data available through the MEL. The software provides a stand-alone decoding application as well as a programmer's interface to the library functions such that GRIB encoding and decoding can be incorporated directly into user applications.

3.2 SOFTWARE INVENTORY

The inventory of all files that make up the MEL GRIB Software Library is listed in **Appendix B**.

3.3 SOFTWARE ENVIRONMENT

The MEL GRIB Software Library 3.0 was developed in ANSI C and should be applicable to any computing environment. However, this software has been developed in the UNIX environment, and all testing has been limited to Silicon Graphics, Inc. (SGI) and Sun platforms. It has not been tested on other platforms/environments.

A C compiler must be installed on the target computer before the MEL GRIB Software Library can be used. The default installation of the Library includes source code files, C include files, and table files, as well as a complete decoding application and examples of simple decoding and encoding programs. Some programming is required (in any language that can interface with C functions) to attain full use of the MEL GRIB Software Library.

3.4 SOFTWARE ORGANIZATION AND OVERVIEW OF OPERATION

The functions in the MEL GRIB Software Library are designed to be called by a user-developed main program. There are many different program organizations for using this Library. **Section 3.4.1** describes the sample encoding process at a high-level, as is depicted by the flow diagram in **Figure 1**. **Section 3.4.2** describes the sample decoding process at a high-level, as is depicted by the flow diagram in **Figure 2**. Actual programs will vary from these examples depending on the structure of the local storage mechanism (i.e., IEEE¹ files vs. relational database).

¹ Institute of Electrical and Electronics Engineers

GRIB USER MANUAL

3.4.1 ENCODING A GRIB MESSAGE

Figure 1 depicts the process used for encoding one or more GRIB messages of the same geometry type. The following steps are followed to encode:

- a. Call `init_enc_struct` to initialize the three internal encoding structures.
- b. Fill the `GEOM_IN` structure either manually or from an external file, if it is available. The external file can be read in by calling `ld_enc_geom`. Refer to “`$GRIB_ENV/data/encoder_ex2.geom`” for the required format.
- c. Fill the `USER_INPUT` structure either manually or from an external file, if it is available. The external file can be read in by calling `ld_enc_config`. Refer to “`$GRIB_ENV/config/encoder.config`” for the required format.
- d. Create storage for a floating point array large enough to support this grid’s dimension. The dimension varies for each geometry type and is computed by multiplying the number of rows by the number of columns. In this example, the same geometry is shared by all GRIB messages so the program will ‘recycle’ this array for the entire run.
- e. Call `init_gribhdr` to create storage for `GRIB_HDR` structure (call *once*).
- f. For each GRIB message to be encoded:
 - (1) Fill in the `DATA_INPUT` structure with the information pertaining to the particular 2-D grid to be encoded. This information can be stored in an external file, which can then be read in by function `ld_enc_ffinfo`. Refer to “`$GRIB_ENV/data/encoder_ex2.info`” for the required format.
 - (2) Fill the data array. Unformatted binary data files that hold the data for the grid can be loaded by calling `ld_enc_ieeeff`. Note that these binary files do not contain the 4-byte Header and Tail as in the FORTRAN IEEE files. Also, they must be $(\#rows * \#cols * sizeof(float))$ bytes long.
 - (3) Call `grib_enc` to encode a GRIB message from the information provided in the three encoder structures and the floating point array. The encoded message is returned in structure `GRIB_HDR`.
 - (4) Call `make_default_grbfn` to create a default filename based on the content of the `DATA_INPUT` and `USER_INPUT` structures. Alternatively, a filename can be manually set using any scheme desired.
 - (5) Call `gribhdr2file` to write the encoded message from `GRIB_HDR` structure out to an external file with the specified name.
- g. Free the floating point array used by the encoder.
- h. Call `free_gribhdr` to release storage of `GRIB_HDR` structure (call *once*).

GRIB USER MANUAL

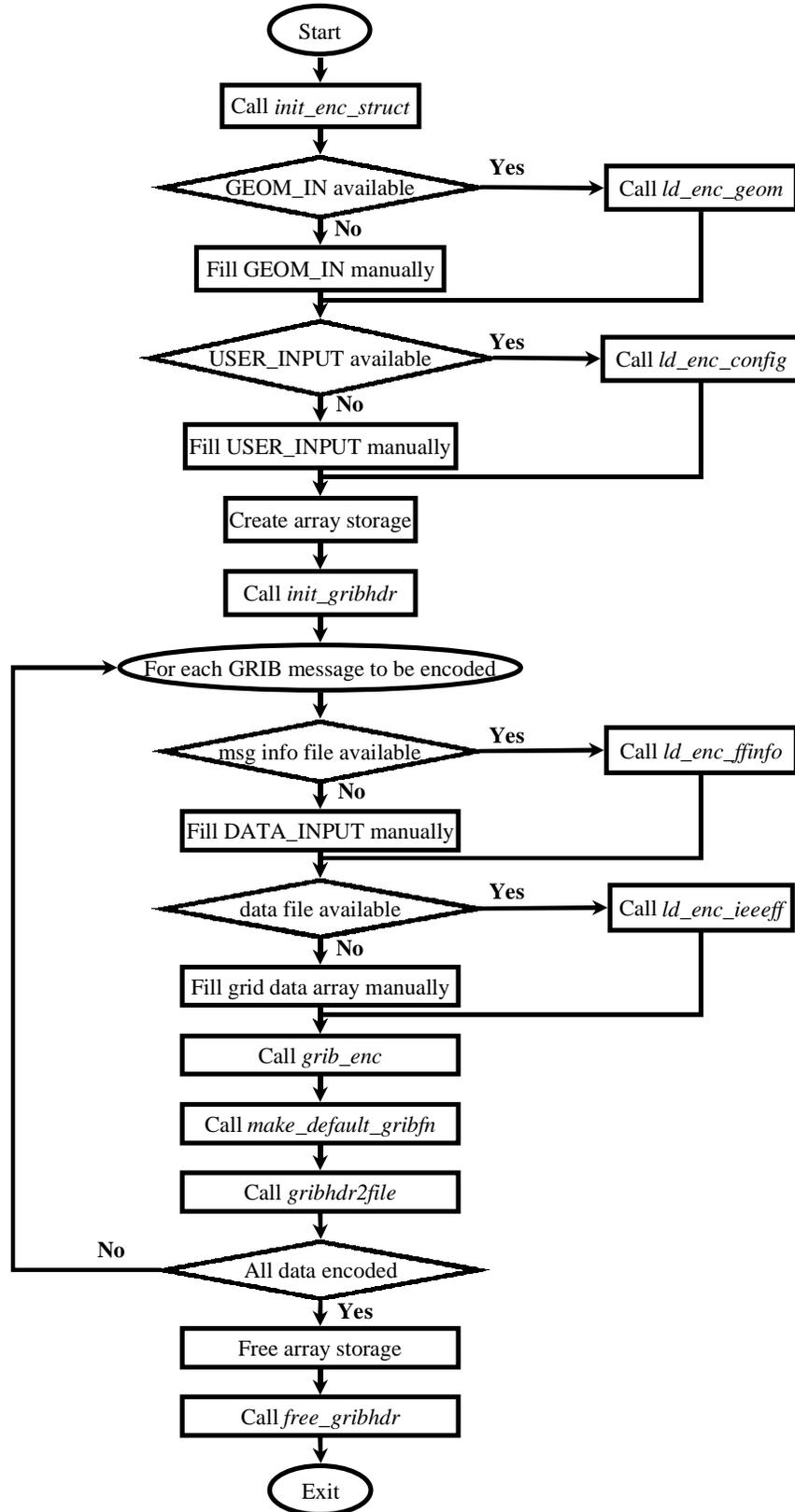


Figure 1. GRIB Encoding Process

GRIB USER MANUAL

3.4.2 DECODING A GRIB MESSAGE

Figure 2 depicts the process for decoding an input file with one or more GRIB messages. The following steps are followed to decode:

- a. Call `init_gribhdr` (*once* at program start) to create GRIB_HDR structure storage.
- b. Open the input file containing GRIB messages and initialize message pointer to 0.
- c. Loop indefinitely to process all GRIB messages in the input file:
 - (1) Call `grib_seek` to load the next message from the input file into GRIB_HDR. Break out of loop if "end of file" OR "error".
 - (2) Call `init_dec_struct` to clear out the decoder structures.
 - (3) Call `grib_dec` to decode the message in GRIB_HDR. Decoded information is returned in decoder structures and a newly allocated floating point data array.

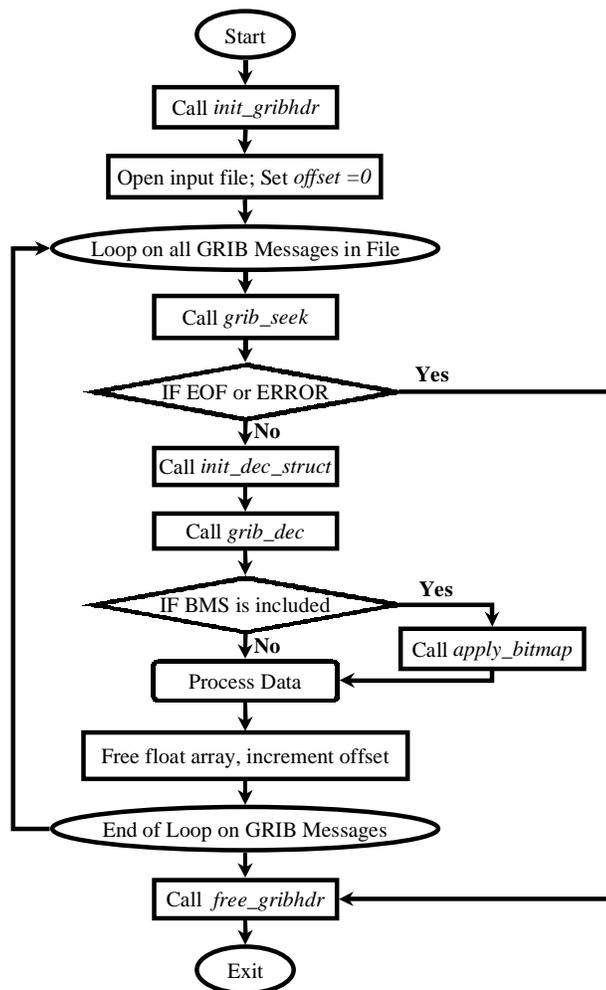


Figure 2. GRIB Decoding Process

GRIB USER MANUAL

- (4) If the Bitmap Section (BMS) is included, call `apply_bitmap` to apply the bitmap to the floating point array to restore the full grid.
 - (5) The float data array can now be processed in any way desired.
 - (6) Free the floating point array created by `grib_dec`
 - (7) Increment the byte offset in the Input File to point to the end of the message just processed. This will tell `grib_seek` where to begin scanning for the next message in the Input File.
- d. End Loop.
- e. Call `free_gribhdr` to release storage for `GRIB_HDR` structure (call once before exiting program).

3.5 CONTINGENCIES/ALTERNATE STATES AND MODES OF OPERATION

Not applicable

3.6 SECURITY AND PRIVACY

3.6.1 US GOVERNMENT SYSTEM

The MEL system and related components are intended for the communication, transmission, processing and storage of US Government information. As such, they are subject to monitoring to ensure proper functioning, protect against improper or unauthorized use or access, verify presence and performance of certain security features or procedures, and other like purposes. Such monitoring may result in the acquisition, recording and analysis of data being communicated, transmitted, processed, or stored in the system. If monitoring reveals evidence of possible criminal activity, the evidence may be provided to law enforcement personnel. Use of the MEL system constitutes consent to such monitoring. The Disclaimer page available via a hyperlink from the MEL Web page describes the security and monitoring agreements for all MEL users.

3.6.2 LIMITS OF LIABILITY

The MEL GRIB Software Library is being furnished without cost and, therefore, "with all faults and as is." The DMSO; Naval Research Laboratory, Marine Meteorology Division; and Science Applications International Corporation do not provide any warranties, expressed or implied, that the software as provided will meet your requirements or that its operation will be uninterrupted or error free.

GRIB USER MANUAL

By making the software available without cost from the DMSO; Naval Research Laboratory, Marine Meteorology Division; and Science Applications International Corporation, there are no implied warranties of merchantability or fitness for a particular purpose nor are there any obligations to provide any software support. There is no warranty that the software is free from defects in materials or workmanship. There is no warranty that the software will perform substantially in accordance with any specifications set forth in the documentation. There is no obligation on the part of the DMSO; Naval Research Laboratory, Marine Meteorology Division; or Science Applications International Corporation to replace any software or hardware on which it may be installed.

The DMSO; Naval Research Laboratory, Marine Meteorology Division; and Science Applications International Corporation shall not be liable for special, incidental, consequential, indirect or other similar damages arising from the use of the software, even if the DMSO; Naval Research Laboratory, Marine Meteorology Division; and Science Applications International Corporation have been advised of the possibility of such damages.

In no event will the DMSO; Naval Research Laboratory, Marine Meteorology Division; and Science Applications International Corporation be liable for loss of data, lost profits, indirect damages arising from the use of the program, however caused and on any theory of liability.

3.6.3 LIMITS OF ENDORSEMENT

References to any commercial products, processes or service by trade name, trademark, manufacturer or otherwise does not necessarily constitute or imply its endorsement, recommendation, and favoring by the US Government, US Navy, DMSO, or NRL.

3.7 PROBLEM REPORTING AND RECOMMENDATIONS

The MEL provides users with e-mail access for technical questions, suggestions, or problems. E-mail hyperlinks are generally available at the bottom of each MEL Web page. An e-mail message containing a complete description of a problem and symptoms may be addressed to:

mel_helps@msosa.dmsomil

General MEL comments or recommendations for enhancements may be sent to the same address.

SECTION 4. ACCESS TO THE SOFTWARE

4.1 FIRST-TIME USER OF THE SOFTWARE

4.1.1 EQUIPMENT FAMILIARIZATION

Not Applicable

4.1.2 ACCESS CONTROL

The MEL GRIB Software Library Web page may be accessed at the following Uniform Resource Locator (URL):

http://mel.dms0.mil/cgi-bin/order_grib

4.1.3 INSTALLATION AND SETUP OF THE MEL GRIB SOFTWARE LIBRARY

4.1.3.1 COPY THE SOURCE CODE

Copy the MEL GRIB Software Library compressed and “tarred” file to the directory where the library is to be installed (e.g., /usr/local). Uncompress the file using the following command²:

```
gunzip grib_v3.0.0.tar.gz
```

To expand the tar file, type the following at the command prompt:

```
tar -xvf grib_v3.0.0.tar
```

<p>NOTE: PC Users can expand the MEL GRIB software library using any PC archive utility such as WinZip. Simply open <code>grib_v3.0.0.tar.gz</code> and select a directory for installation.</p>

The software source code will be expanded into a directory structure relative to the current directory, with a main level directory name of `grib_v3.0` (note the patch level is dropped). The following directory structure relative to the current directory should now exist:

² This assumes the user has the GNU Zip file compression application installed on the computer

GRIB USER MANUAL

```
./grib_v3.0
bin      : executable code for programs under src
config  : user configuration files for encoder
data    : sample data used by the encoder/decoder examples
doc     : documentation
include : required include files
lib     : compiled GRIB library
libsrc  : GRIB encoder and decoder library functions
run     : recommended location where to run the example programs
src     : location of included software examples using the
         library
+ decoder_ex : source code for sample decoding programs
+ encoder_ex : source code for sample encoding programs
+ getgribieee: source code for a GRIB to IEEE decoder
+ gribsimp: source code for the MEL decoder program
tables  : files containing GRIB conversion information
```

4.1.3.2 SETTING THE GRIB ENVIRONMENT

The GRIB software requires that the environment variable `GRIB_ENV` be set to the complete base path of the GRIB library structure. For example, if the software is installed under the path `/usr/local`, the variable `GRIB_ENV` should be set to `/usr/local/grib_v3.0`. This is accomplished by typing the following at the command prompt:

```
setenv GRIB_ENV /usr/local/grib_v3.0
```

It is also convenient to include the GRIB "bin" directory in the default path. If necessary, ask the local System Administrator for assistance in modifying the default path of the environment for the computer where the GRIB software will be run.

NOTE: PC Users can add the following lines to the `AUTOEXEC.BAT` file to properly set up the computer environment. The following example assumes the GRIB software has been installed at the root level of `C:\`.

```
SET GRIB_ENV=C:\grib_v3.0
SET PATH=%PATH%;C:\grib_v3.0\bin
```

GRIB USER MANUAL

4.1.3.3 BUILDING THE LIBRARY

When the `make` utility is used to compile and install the MEL GRIB Software Library and all included programs, all of the required system specific environment variables are provided by the file `$GRIB_ENV/config.os`. The following configurations are currently supported by this software version: SGI IRIX 6.2 using the 'cc' compiler, SGI IRIX 6.2 using the 'cc' compiler with Vis5D extensions, SunOS using the `acc` compiler, and SunOS using the GNU C compiler. If the target system configuration does not conform to any of these, follow the instructions under "Creating your Own Operating System Configuration" at the end of '`config.os`' to create a new section for the specific configuration.

If the Vis5D extensions are to be included in the program '`gribsimp`', enter the complete path to the Vis5D source and include files installed on the target system by defining the `V5DSRCPATH` and `V5DINCPATH` variables, respectively, in `GRIB_ENV/config.os`. These two variables are left undefined if the Vis5D extensions are not used. These two variables are only accessed by the `gribsimp` Makefile and do not affect the library build. This install requires that Vis5D is already installed on the target system. This version was developed and tested using Vis5D 4.3, with `binio.c/.h` and `v5d.c/.h` source and include files linked.

NOTE: The names of these files in other versions of Vis5D cannot be guaranteed.

The simplest way to install the GRIB software is to run the 'Install' script provided in the main directory, as defined by the `GRIB_ENV` variable. The 'Install' script requires an argument that instructs the `make` utility which system configuration in '`config.os`' to use. Choose one of the following:

Table 1 . Choosing System Configuration during Installation

<i>Action</i>	<i>Configuration</i>
Install <code>sgi_cc</code> <cr>	SGI (IP28) IRIX64 v6.2
Install <code>v5d_sgi_cc</code> <cr>	SGI (IP28) IRIX64 v6.2, with vis5d
Install <code>sun_acc</code> <cr>	SunOS (sun4c) v4.1.3 (acc v.SC3.0.1)
Install <code>sun_gcc</code> <cr>	SunOS (sun4c) version 4.1.3 (v2.6)
Install <code>XXXX</code> <cr>	where <code>XXXX</code> is the name used in ' <code>config.os</code> ' for your newly defined configuration

GRIB USER MANUAL

NOTE: The library has only been tested on 32-bit architecture computers. There is a `WORD_BIT_CNT` variable that is set in `include/grib.h` that is intended to configure the low-level routines for 16- or 64-bit architectures, but this has not been tested.

4.2 INITIATING A SESSION

The MEL GRIB Software Library is a series of functions that can be called from another program. Initiating a session will depend upon how the user has developed the calling program.

Once the MEL GRIB Software Library has been successfully installed and tested, the system is ready to begin encoding and decoding GRIB messages. The following are some of the common starting points for users of this software:

- **Retrieving data after MEL delivery**

Verify that the program `gribsimp` has been built and is located in the `$GRIB_ENV/bin` directory. Typing and entering `gribsimp` with no arguments provides command line help. The tape archive (tar) file delivered by MEL can be used directly as the input file to `gribsimp`. For further help with `gribsimp`, see **Section 5.3**.

- **Reading GRIB messages directly into an existing software application**

Begin by reading through this Manual, in particular **Section 5.4**, and then working through the examples provided in the directory `$GRIB_ENV/src/decoder_ex`. The `gribsimp` program, located in `$GRIB_ENV/src/gribsimp`, also provides an advanced example of a GRIB decoder.

- **Delivering data in GRIB**

Encoding GRIB is more difficult than decoding it, and requires some understanding of the GRIB format. Start by reading **Section 5.5** and looking over the *WMO Manual 306* (see **Section 2.2**), then work through the examples provided in `$GRIB_ENV/src/encoder_ex`.

4.3 STOPPING AND SUSPENDING WORK

Not Applicable

SECTION 5. PROCESSING REFERENCE GUIDE

5.1 CAPABILITIES

This section provides an overview of the features and limitations of the MEL GRIB Software Library implementation of the WMO GRIB encoder/decoder algorithms. The MEL GRIB Software Library includes some additional descriptors in the GRIB header sections, and does not fully support all features of the WMO *GRIB Edition 1* standard. However, messages generated by this library are fully compliant *GRIB Edition 1* messages.

5.1.1 FEATURES

5.1.1.1 LIBRARY API

An important feature of the MEL GRIB Software Library is that it provides an interface to generic encoding and decoding functions that permit the development of customized GRIB applications and/or the development of a GRIB Input/Output layer for existing applications. The Library is written in C, but it has been successfully incorporated into FORTRAN applications. For more information on the FORTRAN-C interface, contact the developers.

5.1.1.2 ADDITIONAL HEADER INFORMATION

The MEL GRIB Software Library supports the use of four additional parameters in the Product Definition Section (PDS) of the GRIB message. An overview of these parameters is provided here, and **Appendix C** contains a detailed discussion of the implementation of these GRIB extensions.

<p>NOTE: These extensions are completely valid under the WMO GRIB specification, and will not cause problems for any properly written GRIB decoder.</p>
--

Reference Seconds – Seconds have been added to the specification of the reference time

Tracking ID – A 2-byte integer that can be used to track the heritage of a message or dataset

Parameter Sub-ID – Five sub-tables of locally defined parameters are supported

GRIB USER MANUAL

Local Table ID – A version identifier has been added for the local tables

5.1.1.3 EXTERNAL TABLES

The use of externally stored tables for the definition of parameters, level types, geometry and model names makes the MEL GRIB Software Library very flexible in its application. To as great an extent as possible, the encoding and decoding functions of the library are independent of the process of "mapping" descriptive information to and from the codes in a message. This permits the writing of more generic applications that can then be used at multiple encoding centers by simply modifying the external tables. A complete description of the format of the external tables with a discussion on local table management is included in **Section 5.6**.

The file name for the external tables is made up of information contained in the message header sections. This allows decoding applications to automatically determine and attempt to load the required table. In addition, since tables from all encoding centers have a unique name, they can be collected at a central File Transfer Protocol (FTP) site, allowing the decoding application to download required tables automatically.

5.1.2 LIMITATIONS

5.1.2.1 GRID DEFINITION SECTION LIMITATION

Only the following grid types are supported by the MEL GRIB Software Library 3.0:

- Latitude/Longitude (Type 0)
- Mercator (Type 1) (Decoding only)
- Lambert Conformal (Type 3)
- Gaussian Latitude/Longitude (Type 4) (Decoding only)
- Polar Stereographic (Type 5)

Rotated and stretched grids are not supported.

5.1.2.2 PACKING METHODS

The library uses the simple packing method for gridded data. Spherical Harmonic coefficients and second order packing are not supported.

5.2 CONVENTIONS

Appendix E defines default naming conventions for all input and output files used by the Library

GRIB USER MANUAL

There are no unique conventions used by MEL GRIB software regarding use of colors in the displays or audible alarms.

5.3 LEARNING GRIB USING `GRIBSIMP`

The program `gribsimp` is included as a standard part of the Library, and serves as the standard GRIB decoding utility for the MEL project. It provides many useful features as outlined in **Section 5.3.1**, and is a good example of how to build a robust GRIB decoding program. If the library was successfully installed, there should be a compiled version of `gribsimp` in the directory `$GRIB_ENV/bin` that is ready for use.

This section has two objectives:

- a. Demonstrate the use of `gribsimp` in handling the decoding of GRIB messages
- b. Provide a hands-on overview of the GRIB standard

5.3.1 OVERVIEW OF `GRIBSIMP`

The `gribsimp` decoder is a stand-alone program for decoding one or more GRIB messages. It is based on the standard GRIB library, but also adds capabilities that are useful in handling GRIB messages. It is designed to handle any number of valid GRIB messages within a single file and will automatically skip over headers between messages. This means that `gribsimp` can process messages concatenated into one file (by `cat` or `tar`) without any problem. It also allows users to create an index of the messages in a file, so individual messages can be selected for extraction.

The output from the program depends upon the specified command line arguments. As a minimum, `gribsimp` will provide a summary of the GRIB messages contained in the input file. Optionally, `gribsimp` can be used to generate IEEE binary data files for each message decoded, create a Vis5D³ data set, or even generate a Grid Analysis and Display System (GrADS)⁴ data set complete with a script to visualize each GRIB message. Each of the `gribsimp` output options will be discussed and explored through examples in the following sections.

³ Available at <http://www.ssec.wisc.edu/~billh/vis5d.html>

⁴ Available at <http://grads.iges.org>

GRIB USER MANUAL

As discussed in **Section 5.6.1**, the GRIB standard relies on code tables to transform descriptive information into binary code. The WMO provides a standard set of parameter and level definitions, but GRIB producers are permitted to add their own definitions, so there is no standard GRIB decoding table that covers all messages. The MEL GRIB Software Library works at the binary code level, and does not try to decipher what is meant by the codes it encounters. This makes the Library more robust and universal, but it leaves the task of deciphering the binary codes to the user. The program `gribsimp` attempts to automate the use of any required decoding tables by determining which decoding table is required using information in each message. It then attempts to load the required table from the local `$GRIB_ENV/tables` directory, and will attempt to download the required table from an FTP site if it is not found locally.

NOTE: The automated FTP feature requires that the software be run on a system that supports the UNIX `sh` environment.

If `gribsimp` cannot find the table it requires, it will prompt the user with options of either using the default WMO table or providing a path to a table they wish to use.

5.3.2 COMMAND LINE OPTIONS

This section provides a complete description of the command line options available for `gribsimp`. The next section will present examples demonstrating how to combine the command line arguments for common decoding operations. The command line syntax for `gribsimp` follows:

```
gribsimp -i infile [-d ] [-o] [-v] [-b] [-I indx] [-print][-g [dtg lvl]] [-v5d dtg
lvl [v5dfn]]
                [-D [path]]           [-X indx] [-dump]
                [-t ]
                [-s Table]
```

-i : Required flag for specifying an input GRIB file, specified by 'infile.' The input file can contain one or more concatenated messages (i.e., combined using `tar`), with or without headers between the messages. The `gribsimp` program will exit if no input file is specified.

-d, -D, -t, -s: `gribsimp` expects only one of these four flags. `-d` instructs `gribsimp` to create a summary log file called `GRIB.log` that contains all header information and the first 100 data points from each message in the input file. `-D` creates the same summary file as `-d`, but performs table lookups that provide text descriptions for the parameter and units, the level, the model, and the geometry, if included. For more information on the format of the decoder table name, see **Appendix E.4**. `gribsimp` builds the table name from header information in each message and then searches for the table in the following three locations:

- a. The directory specified by the optional 'path' parameter to `-D`

GRIB USER MANUAL

- b. The `$GRIB_ENV/tables` directory
- c. The directory specified by the `TABLES_PATH` variable declared at the top of `gribsimp.c` (default value is `../tables`).

If `gribsimp` is unable to find the required table in any of these locations, it will attempt to download the required table from the MEL GRIB Tables FTP site using a UNIX shell script. (**Note:** This feature requires the software runs on a system that supports the UNIX sh environment). `gribsimp` will abort if it is unable to locate the required table, or the table it uses does not contain a definition for one of the binary codes in a message. The user then has three options to choose from before proceeding with decoding:

- a. The user is urged to contact the activity that generated the message to obtain the proper table
 - b. The `-s` option can be used to instruct `gribsimp` to use a specified table. The specified table must be located in the `$GRIB_ENV/tables` directory.
 - c. The `-t` option can be used to instruct `gribsimp` to use the default WMO decoder table that contains only the WMO parameter and level definitions (**Note:** This table contains no model or geometry definitions).
- `-o:` Instructs `gribsimp` to generate a 32-bit IEEE unformatted binary data file for each message decoded. These binary files do *not* contain the 4-byte header or trailer as in the IEEE files created in FORTRAN. They contain just the floating-point data, organized as is specified in the Grid Definition Section of the message.
- The file naming convention and format for the files generated by this option is described in **Appendix E.3**.
- `-b:` Instructs `gribsimp` to not apply any bitmap found to the data. This means `gribsimp` will only write out the defined data points and it will be left to the user to accurately apply a bitmap, if required.

CAUTION: The `-b` option should only be used when the user is *very* familiar with the data being decoded.

- `-v:` Displays the version information for `gribsimp`. This option is useful when seeking help with a decoding problem.
- `-I:` Instructs `gribsimp` to generate an index file whose name must be specified after the `-I`. The index file contains one descriptive line for each valid GRIB message found in 'infile.' The index file can then be reviewed and any messages the user does not wish to decode may be deleted.

GRIB USER MANUAL

The first line of the index file contains header information describing the format of the subsequent line and should never be deleted from the file. The remaining lines each describe a single valid GRIB message, using the following format:

YY-MM-DD-HH-TAU-PID-SUB-LVL-HEIGHT-GID-OFFSET

This represents the Year, Month, Day, and Hour of the Reference Time, followed by the Forecast Period, the Parameter ID, the Parameter Sub-ID, the Level ID, the Height value, the Grid ID, and the byte location where this message begins within the input file.

- X: This option instructs `gribsimp` to decode only those messages that are listed in an index file previously generated using the `-I` option. The index file name must be specified after the `-X` and must be used with the same input file used to create it.

NOTE: When in this mode, `gribsimp` will not scan the input file for valid GRIB messages, but rather just decode the messages at the byte locations specified in the index file. Therefore, it is extremely important that the byte location values not be modified when editing the index file.

- print: Prints the internal GRIB structures to the screen after decoding each message. This option is useful for solving decoding problems with messages from a new source.
- dump: Prints as much information as possible from an incomplete message. This option is useful for finding out how much information is actually valid in an incomplete message.
- g: Creates a GrADS data set from the GRIB messages in 'infile.' The output will be the required GrADS `.gmp` and `.ctl` files, and a script called `draw_all.gs` that will display an animated sequence of all messages included in the GrADS data set. The default uses the reference time and level ID of the first message as the basis of the data set. An alternate date/time (of form `yyyymmddhh`) and level ID may be specified explicitly. Refer to Examples 4 and 5 in **Section 5.3.3** for more information. The GrADS software and documentation are available at <http://grads.iges.org>

NOTE: The GrADS option only supports spherical and Lambert conformal grid definitions.

GRIB USER MANUAL

- v5d : Creates a Vis5D data set starting from the specified date and time using the specified level type for the vertical dimension. Both the date-time group and the GRIB level type indicator are required parameters. Optionally, an output file name can also be specified as the third parameter. If no file name is specified, a default file name of the format 'yyyymmddhh.lvl.v5d' will be used, where yyyymmddhh is the date-time group and 'lvl' is the level type ID. Refer to Example 6 in **Section 5.3.3** for more information.

There are only two GRIB level types supported at this time:

- 100 — Implies input data is expressed as pressure levels.
- 105 — Implies input data is expressed as constant height above ground level

Only one level type can be specified, and a non-valid level type ID will cause `gribsimp` to exit. Note that for level type 105, `gribsimp` will also load any surface (001) and/or mean sea level (102) messages in the input file as constant height surfaces (105) with a value of 0. If there are no messages in the input file that match the specified date-time and level type, no output file will be created.

The Vis5D option is ONLY available if `gribsimp` was compiled with the Vis5D environment variables defined, and requires that Vis5D be installed on your system. Refer to **Section 4.1.3.3** for installation instructions. Refer to the Vis5D documentation⁵ for information on obtaining and using Vis5D.

The Vis5D option is not fully implemented in this version of the GRIB software. The following assumptions are made in creating Vis5D data sets:

- Only spherical and Lambert conformal grid projections are supported
- Only defines text names for a handful of parameters. All others have a default name built from the GRIB parameter code.
- Assumes all messages in the input file are for the same model and geometry. Will drop any messages that do not match the model and geometry of the first message.
- Will only process messages of same or later date-time than that specified on the command line.

⁵ Available at <http://www.ssec.wisc.edu/~billh/vis5d.html>

GRIB USER MANUAL

5.3.3 EXAMPLES USING GRIBSIMP

This section includes a number of examples to illustrate features of `gribsimp`, and should also help to understand the organization of GRIB messages. Refer to **Appendix A** of this manual and *WMO Manual 306: Manual on Codes* for details on the GRIB standard. In each example the command to be executed is presented first, followed by the expected result that should be seen on the display.

All of the examples use the sample GRIB data file `GRIB_ENV/data/GRIB0797.tar`, which is a tar file containing six GRIB messages.

5.3.3.1 EXAMPLE 1: DECODING YOUR FIRST MESSAGE

At the command prompt, type the following:

```
gribsimp -i GRIB_ENV/data/GRIB0797.tar -d -v
```

```
GRIBSIMP Execution...
Software release: 3.0
GRIB standard version: 1
Grib Library in non-verbose mode
Decoding message found at 512 bytes ...
Decoding message found at 4608 bytes ...
Decoding message found at 8192 bytes ...
Decoding message found at 12288 bytes ...
Decoding message found at 16384 bytes ...
Decoding message found at 20480 bytes ...
```

This command performs the basic decoding operation on the specified input file, and includes the `-v` option to obtain version information about the decoder. The `-i` option is used to specify an input data file, and the `-d` option will generate a log file (`GRIB.log`) containing header information and a sample of data from each message decoded. With the `-d` option, no attempt is made to load a decoder table and decipher the codes used in the header section. Since the use of decoder tables is often a source of error, it is a good idea to first decode the messages with only the `-d` option to ensure the input file contains valid GRIB messages. Compare the `GRIB.log` file created with the file `GRIB_ENV/data/GRIB.test1` to verify that the library and `gribsimp` are working correctly.

5.3.3.2 EXAMPLE 2: ADDING INDEXING AND ADDITIONAL PRINTING

At the command prompt type the following:

```
gribsimp -i GRIB_ENV/data/GRIB0797.tar -D -I index
```

```
GRIBSIMP Execution...
```

GRIB USER MANUAL

```
Decoding message found at 512 bytes ...
-> LookupTbl '$GRIB_ENV/tables/gltab_58_2.0'
Decoding message found at 4608 bytes ...
Decoding message found at 8192 bytes ...
Decoding message found at 12288 bytes ...
Decoding message found at 16384 bytes ...
Decoding message found at 20480 bytes ...
```

This time the `-D` option has been used to instruct `gribsimp` to load a decoder table based on the information in the message, and decipher the binary codes for parameter name, level type, model and geometry name. More descriptive information is then printed to `GRIB.log` to help interpret the data contained in the messages. Compare `GRIB.log` to `$GRIB_ENV/data/GRIB.test2` to make sure `gribsimp` is locating the decoder tables correctly. The format and contents of a decoder table file is described in **Section 5.6.1.1**.

Also note the use of the `-I` option in this example. This creates an index file, called 'index' in this case, containing a summary of the messages in the specified input file. The contents of this index file are shown below:

```
YY-MM-DD-HR-Tau-ParmID-ParmSubID-LevelID-Level-GDS_Data_Type-Offset
97-07-01-00-000-002-000-102-00000-237-512
97-07-01-00-012-002-000-102-00000-237-4608
97-07-01-00-000-011-000-105-00002-237-8192
97-07-01-00-012-011-000-105-00002-237-12288
97-07-01-00-000-033-000-105-00010-237-16384
97-07-01-00-012-033-000-105-00010-237-20480
```

For each message, a one line record has been written to the index file that consists of ten parameters which uniquely specify a product, (for a given model type), plus an offset that indicates the byte position of the start of that message in the input file. This file can then be used to select individual messages to decode in any order. The index file entries are discussed in detail in **Section 5.3.2** under options `-I` and `-X`.

5.3.3.3 EXAMPLE 3: CONTROLLING THE DECODING PROCESS

In this example, decoding is restricted to a selected subset of the messages in the input file, and `gribsimp` is instructed to write binary files containing the data for each message decoded. Begin by editing the file 'index' from the last example so that only the first, third, and fifth messages remain in the file. The index file should now look like this:

```
YY-MM-DD-HR-Tau-ParmID-ParmSubID-LevelID-Level-GDS_Data_Type-Offset
97-07-01-00-000-002-000-102-00000-237-512
97-07-01-00-000-011-000-105-00002-237-8192
97-07-01-00-000-033-000-105-00010-237-16384
```

At the command prompt type the following:

```
gribsimp -i $GRIB_ENV/data/GRIB0797.tar -X index -o
```

GRIB USER MANUAL

```
GRIBSIMP Execution...
```

```
Decoding message found at 512 bytes ...  
Creating flat file= 'FF97070100000002.237.102.00000'  
Decoding message found at 8192 bytes ...  
Creating flat file= 'FF97070100000011.237.105.00002'  
Decoding message found at 16384 bytes ...  
Creating flat file= 'FF97070100000033.237.105.00010'
```

Note that this time `gribsimp` has only processed the three messages selected in the index file — the three messages deleted were not processed even though they still remain in the input file. The `-o` option was used to create IEEE 32-bit unformatted binary files for each message decoded. The storage order of the data in the output files is specified in the Grid Definition Section of each message (which is in the `GRIB.log` file created in Example 1).

For more information on the structure and naming convention used for the files generated by the `-o` option, see **Section 5.3.2** and **Appendices E.2 and E.3**.

5.3.3.4 EXAMPLE 4: CREATING A GRADS DATA SET

The option `-g [dtg lvl]` can be included with the `gribsimp` call to create the control files required for input to the GrADS public domain visualization system. GrADS defines a five-dimensional (5-D) data set that includes three spatial dimensions, time, and a parameter list. When the `-g` option is used without the date-time group (`dtg`) and level type (`lvl`) arguments `gribsimp` will create a GrADS data set based on the date-time group and level type of the first valid message found in the input file. The specification of the reference date-time group and level type is very important because `gribsimp` makes the following assumptions in building a GrADS data set:

- a. Only messages with a reference time equal to or later than the specified date-time group will be included in the data set.
- b. Only messages with the level type specified will be included in the three-dimensional (3-D) volume defined for the GrADS data set. Messages of other level types will be treated as single level fields in the data set. Refer to the GrADS documentation for more information on how GrADS treats data sets with multiple vertical coordinate systems.

In this example, no arguments are specified with the `-g` option so `gribsimp` will use the date-time group and level type of the first message in the input file. At the command prompt, type the following:

```
gribsimp -i $GRIB_ENV/data/GRIB0797.tar -g
```

```
GRIBSIMP Execution...
```

```
Decoding message found at 512 bytes ...  
Decoding message found at 4608 bytes ...
```

GRIB USER MANUAL

```
Decoding message found at 8192 bytes ...
Decoding message found at 12288 bytes ...
Decoding message found at 16384 bytes ...
Decoding message found at 20480 bytes ...
GrADS Control Files= 'GRIB0797.ct1' & 'GRIB0797.gmp'
* Warning: Zdef level (102) only has one Height (00000)
```

This creates the following three output files: a GrADS control file describing this data set, `GRIB0797.ct1`, the associated GRIB mapping file, `GRIB0797.gmp`, and a GrADS script, `draw_all.gs`, that can be called from within the GrADS program. Since no `dtg` or level arguments were given, the first message's level (102) and `dtg` were chosen as the default values.

NOTE: The last line is a warning to the user that Level 102 only has one height, which in this case is 00000. To accommodate GrADS when this happens, `gribsimp` adds an additional height value to the data set.

The GrADS data set is defined by the control (`.ct1`) file, which is shown below for this data set. Note that the input data file specified in the GrADS control file is the tar file itself. GrADS can read GRIB files directly once the GRIB mapping file (`.gmp`) is generated, which makes it a very convenient and efficient visualization program for GRIB data.

`gribsimp` has created three GrADS variables based on their parameter and level ID's in the messages. This is done to keep `gribsimp` independent of the local table definitions. In the next example, the `-D` option will be used, and the GrADS variables definitions will include the descriptions contained in the decoding table for this data set.

Parameters defined on the level type specified in the ZDEF statement have the variable name 'aPidLid', where 'Pid' is the 3-digit Parameter ID and 'Lid' is the 3-digit Level ID. Parameters defined on a level type different than the ZDEF level type have an additional character appended to their variable name to maintain uniqueness.

GRIB0797.ct1:

```
dset ^/a/nakajima/GRIB/grib/data/GRIB0797.tar
dtype grib
index ^GRIB0797.gmp
undef -9.99E+33
title /a/nakajima/GRIB/grib/data/GRIB0797.data
* pdef isz jsz LCC reflat reflon iref jref stdlat1 stdlat2
stdlon delx dely
xdef 61 LINEAR 126 0.200
ydef 51 LINEAR 29 0.200
* Level (102) only has 1 Height (00000), add dummy 2nd Height
zdef 2 levels
```

GRIB USER MANUAL

```
00000 00001
tdef      2 linear 00Z01jul97 12hr
vars 3
a002102  1 002,102      [No Lookup File]
a011105a 0 011,105,002 [No Lookup File]
a033105a 0 033,105,010 [No Lookup File]
endvars
```

To demonstrate the result of the GrADS option, the script 'draw_all.gs' is created which displays each parameter from the first to last height for each time increment. Invoke GrADS by typing:

NOTE: This procedure assumes GrADS is installed on the host computer.

```
grads -l
Grid Analysis and Display System (GrADS) Version 1.5 Beta-Final
Copyright (c) 1988-1994 by Brian Doty
Center for Ocean-Land-Atmosphere Studies
Institute for Global Environment and Society
All Rights Reserved
```

```
GX Package Initialization: Size = 11 8.5
ga>
```

At the GrADS prompt (ga>), type `run draw_all.gs`. GrADS will display the first field in the data set and prompt the user to press the <Enter> key. Each time <Enter> is pressed, GrADS will display the next field. In this GrADS session, six grids are plotted in the following order:

- a. Pressure (a002102) at Level 102 Height 00000 for Time 1 (00Z)
- b. Pressure (a002102) at Level 102 Height 00000 for Time 2 (12)
- c. Temperature (a011105a) at Level 105 Height 002 for Time 1 (00Z)
- d. Temperature (a011105a) at Level 105 Height 002 for Time 1 (00Z)
- e. U-Wind (A033105a) at Level 105 Height 010 for Time 1 (00Z)
- f. U-Wind (A033105a) at Level 105 Height 010 for Time 1 (00Z)

GrADS Session:

```
ga> run draw_all.gs
ZDEF Level has 1 heights,  Z = { 00000; }
GRIB0797.ct1 [Varname #enumerated Parm,Lvl      ] ->
==> a002102  1 002,102      [No Lookup File]
draw a002102 at Z=1 for (T=1,2)
Press enter to continue:::::
-----
GRIB0797.ct1 [Varname #enumerated Parm,Lvl,HEIGHT ]
=> a011105a  0 011,105,002 [No Lookup File]
```

GRIB USER MANUAL

```
drawing a011105a at LEV=2 (T=1,2)
Press enter to continue:::::
-----
GRIB0797.ct1 [Varname #enumerated Parm,Lvl,HEIGHT ]
=> a033105a 0 033,105,010 [No Lookup File]
drawing a033105a at LEV=10 (T=1,2)
Press enter to continue:::::
-----
ga>
```

5.3.3.5 EXAMPLE 5: A SECOND GRADS EXAMPLE

In this example, a specific date-time group and level ID are specified with the `-g` option to `gribsimp`. This is useful if the messages in a GRIB data set are not ordered chronologically, or if the first message does not contain the level type to be used for the 3-D volume of data. The `dtg` has the format of `YYYYMMDDHH`, and should be defined as the earliest reference time desired in the data set. The level ID specified must be contained in at least one of the messages in the data set. Specifying arguments with the `-g` option requires some knowledge about the data set to be processed, but this information may be determined by using the options discussed in the previous examples.

This example also uses the `-D` option, which instructs `gribsimp` to look up descriptive information for the variables to be included in the GrADS data set.

In the previous example the input file contained six GRIB messages: parameters Pressure, Air Temperature, and Wind U-Component at two forecast times relative to the base date-time group. The Pressure fields are defined on a level type of “Mean Sea Level” (Level ID 102), while the other two parameters are defined on a level type of “Constant Height above Ground” (Level ID 105). In this example, the level type of 105 is specified on the command line, instructing `gribsimp` to build the GrADS ZDEF statement based on level type 105 instead of using the level type of the first message it processes, 102.

At the command prompt, type the following:

```
gribsimp -i $GRIB_ENV/data/GRIB0797.tar -D -g 1997070100 105
```

```
GRIBSIMP Execution...
```

```
Decoding message found at 512 bytes ...
-> LookupTbl '$GRIB_ENV/tables/gltab_58_2.0'
Decoding message found at 4608 bytes ...
Decoding message found at 8192 bytes ...
Decoding message found at 12288 bytes ...
Decoding message found at 16384 bytes ...
Decoding message found at 20480 bytes ...
GrADS Control Files= 'GRIB0797.ct1' & 'GRIB0797.gmp'
```

GRIB USER MANUAL

Notice that in the resulting control file (shown below) the ZDEF statement has changed from the previous example and now contains the two heights found in the input data set. This results in a slightly different set of variable names as well, because now the Air Temperature (011) and Wind U-Component (033) fields are treated as part of the 3-D volume defined by ZDEF instead of individual special levels.

The variable definition lines now also contain a descriptive comment for each parameter. This is a result of the `-D` option used, and these comments are taken from the decoder table specific to this data set.

```
GRIB0797.ct1:
dset ^/a/nakajima/GRIB/grib/data/GRIB0797.tar
dtype grib
index ^GRIB0797.gmp
undef -9.99E+33
title /a/nakajima/GRIB/grib/data/GRIB0797.data
* pdef isz jsz LCC reflat reflon iref jref stdlat1 stdlat2
stdlon delx dely
xdef      61 LINEAR      126 0.200
ydef      51 LINEAR      29 0.200
zdef 2 levels
00002 00010
tdef      2 linear 00Z01jul97 12hr
vars 3
a002102a  0 002,102,000 pressure reduced to msl [Pa]
a011105   2 011,105     temperature [K]
a033105   2 033,105     wind u-component [m/s]
endvars
```

The 'draw_all.gs' script will now attempt to display the following fields:

- a. Pressure field at level 102 Height 000 for 00Z
- b. Pressure field at level 102 Height 000 for 12Z
- c. Temperature field at level 105 Height 002 for 00Z
- d. Temperature field at level 105 Height 002 for 12Z
- e. Temperature field at level 105 Height 010 for 00Z (does not exist)
- f. Temperature field at level 105 Height 010 for 12Z (does not exist)
- g. Wind-U component field at level 105 Height 002 for 00Z (does not exist)
- h. Wind-U component field at level 105 Height 002 for 12Z (does not exist)
- i. Wind-U component field at level 105 Height 010 for 00Z
- j. Wind-U component field at level 105 Height 010 for 12Z

GRIB USER MANUAL

Notice that because the level used for the ZDEF is now 105, the script attempts to draw the Temperature and Wind fields at both Heights 002 and 010 even though the Temperature only exists at Height 002 and the Wind only exists at Height 010. This results in the GrADS error message 'Cannot contour grid - all undefined values' when attempting to draw those grids.

GrADS session:

```
grads -l
Grid Analysis and Display System (GrADS) Version 1.5 Beta-Final
Copyright (c) 1988-1994 by Brian Doty
Center for Ocean-Land-Atmosphere Studies
Institute for Global Environment and Society
All Rights Reserved
```

```
GX Package Initialization: Size = 11 8.5
```

```
ga> run draw_all.gs
```

```
GRIB0797.ct1 [Varname #enumerated Parm,Lvl,HEIGHT ]
=> a002102a 0 002,102,000 pressure reduced to msl [Pa]
drawing a002102a at LEV=0 (T=1,2)
Press enter to continue:::::
```

```
-----
ZDEF Level has 2 heights, Z = { 00002; 00010; }
GRIB0797.ct1 [Varname #enumerated Parm,Lvl ] ->
==> a011105 2 011,105 temperature [K]
draw a011105 at Z=1 for (T=1,2)
Press enter to continue:::::
GRIB0797.ct1 [Varname #enumerated Parm,Lvl ] ->
==> a011105 2 011,105 temperature [K]
draw a011105 at Z=2 for (T=1,2)
Cannot contour grid - all undefined values
Cannot contour grid - all undefined values
Press enter to continue:::::
```

```
-----
ZDEF Level has 2 heights, Z = { 00002; 00010; }
GRIB0797.ct1 [Varname #enumerated Parm,Lvl ] ->
==> a033105 2 033,105 wind u-component [m/s]
draw a033105 at Z=1 for (T=1,2)
Cannot contour grid - all undefined values
Cannot contour grid - all undefined values
Press enter to continue:::::
GRIB0797.ct1 [Varname #enumerated Parm,Lvl ] ->
==> a033105 2 033,105 wind u-component [m/s]
draw a033105 at Z=2 for (T=1,2)
Press enter to continue:::::
```

```
-----
ga>
```

GRIB USER MANUAL

5.3.3.6 EXAMPLE 6: CREATING A VIS5D DATA SET

The `-v5d` option can be included with the `gribsimp` call to create a Vis5D data set. Vis5D defines a 5-D data set that includes three spatial dimensions, time, and a parameter list. The `-v5d` option requires that a date-time group and a level type ID (lvl) be provided. The resulting Vis5D data set will contain all messages from the input file that match the level type specified, and that are at or later than the specified date-time.

As previously mentioned, the `-v5d` option is only available if `gribsimp` was compiled with the Vis5D extensions, and requires that the Vis5D system be installed on the target system.

At the command prompt, type the following:

```
gribsimp -i $GRIB_ENV/data/GRIB0797.tar -v5d 1997070100 105
```

```
GRIBSIMP Execution...
vis5d will process Levels 001 & 102 as Level 105 with Ht=0
Decoding message found at 512 bytes ...
V5DMSG: 97070100 tau=000, Parm=002 Sub=000 Lvl=102 (Ht=00000)
Decoding message found at 4608 bytes ...
V5DMSG: 97070100 tau=012, Parm=002 Sub=000 Lvl=102 (Ht=00000)
Decoding message found at 8192 bytes ...
V5DMSG: 97070100 tau=000, Parm=011 Sub=000 Lvl=105 (Ht=00002)
Decoding message found at 12288 bytes ...
V5DMSG: 97070100 tau=012, Parm=011 Sub=000 Lvl=105 (Ht=00002)
Decoding message found at 16384 bytes ...
V5DMSG: 97070100 tau=000, Parm=033 Sub=000 Lvl=105 (Ht=00010)
Decoding message found at 20480 bytes ...
V5DMSG: 97070100 tau=012, Parm=033 Sub=000 Lvl=105 (Ht=00010)

Entering make_v5d_file:
Time#1 (241032.000000) is 1997/07/01 hr= 0.000000 => '97182' '000000'
Time#2 (241044.000000) is 1997/07/01 hr= 12.000000 => '97182' '120000'
Height[3]= ( 00000; 00002; 00010;)
Vis5d summary:
PRES_MSL P002S000 has Heights [00000 (ht#0) - 00000 (ht#0)] so Nht=1
AIR_TEMP P011S000 has Heights [00002 (ht#1) - 00002 (ht#1)] so Nht=1
U P033S000 has Heights [00010 (ht#2) - 00010 (ht#2)] so Nht=1
Vis5d file created= '1997070100.105.v5d'
```

The resulting output file '1997070100.105.v5d' is ready to be loaded and viewed with the Vis5D program. Refer to the Vis5D documentation for information on using Vis5D to visualize data.

5.4 DECODING EXAMPLE

The previous section described how to decode and manipulate GRIB messages using the stand-alone application, `gribsimp`. The program `decoder_ex.c` provides a starting point for users who want to write custom decoding programs or include calls to the GRIB decoding functions from an existing application.

GRIB USER MANUAL

The program `getgribieee.c` is another decoder example showing how to extract just the data portion of the GRIB message and store it in external IEEE data files.

For more information on how to run these programs, refer to **Appendix D**.

5.4.1 PROGRAM 'DECODER_EX.C'

`decoder_ex.c` is an example program showing how one can decode an input file that has multiple-GRIB messages by calling the MEL GRIB Software Library functions directly. The input file used is `$GRIB_ENV/data/GRIB0797.tar`, which contains six GRIB messages.

The MEL GRIB Software Library functions are used as follows to create a simple decoding program:

```
init_gribhdr() /* to create storage for GRIB_HDR structure (do only once) */
Loop {
    grib_seek() /* to load next message in input file into GRIB_HDR */
    init_dec_struct() /*to clear out the decoder structs */
    grib_dec () /*to decode message in GRIB_HDR*/
    /* if successful, float data is returned in a newly allocated array and message info is
    returned in decoder structures. */
    apply_bitmap() /* only if a Bit Map Section (BMS) is present */
    /* free float array allocated by grib_dec here */
} /*close Loop */
free_gribhdr() /*to release storage for GRIB_HDR structure. */
```

Upon each successful call to `grib_seek`, a new message will be stored in the `GRIB_HDR` structure, along with pointers to each section and section lengths for the message. After `grib_dec` is called with `GRIB_HDR` as input, message information is stored in the decoder structures `PDS_INPUT`, `grid_desc_sec`, `BMS_INPUT`, `BDS_HEAD_INPUT`, and the float data is returned in an array allocated within `grib_dec`. The data may be used in any manner at this point. Before looping back to search for the next message the array containing the float data should be freed. The function `free_gribhdr()` will cleanly destroy `GRIB_HDR`, and should always be called prior to exiting the program.

To process all of the messages in an input file, continue looping until `grib_seek` returns the 'no more message' status of 2, or an error occurs. On the first call to `grib_seek`, the offset should be set to zero so that searching begins at the start of the input file. Prior to subsequent calls to `grib_seek`, offset should be incremented by the length of the message just processed. Failure to increment the offset could result in an infinite loop.

Explanation of 'decoder_ex':

- a. `gribfuncs.h` needs to be included since it contains the function prototypes

GRIB USER MANUAL

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "gribfuncs.h" /* GRIB library function prototypes
*/
#define TABLE_PATH      "../tables"      /* default dir of
Lookup Tables */
#define FILL_VALUE       -9999.00        /* default for
missing data pts */

/* Main Program Starts Here */
void main (int argc, char **argv) {
```

b. Define the variables needed

```
FILE *fout;          /* output file */
char InFile[200];    /* input file name */
char *grib_env_dir; /* holds value of GRIB_ENV environment
variable */
char errmsg[2000];   /* required, error msgs back from Grib
library */
char OutFn[200];     /* name of output file */
int i;               /* working var */
int D;               /* Decimal Scale Factor */
int decoded_cnt;     /* number of messages decoded
successfully */
int nReturn=0;       /* return status from Grib_dec */
int Rd_Indexfile=0; /* Zero to decode all msgs in input
file*/
long offset; /* byte offset from beginning of file, for
input files with multiple GRIB messages; */
float *grib_data;    /* pointer to block of decoded data */
BMS_INPUT bms;      /* structure for bitmap section */
PDS_INPUT pds;      /* structure for product definition section
*/
grid_desc_sec gds; /* structure for grid description
section */
BDS_HEAD_INPUT bds_head; /* structure for bds section */
GRIB_HDR *gh1;      /* POINTER to the grib header
structure; actual storage is allocated
by the call to init_gribhdr() and is
released by free_gribhdr(). Used to hold
the GRIB message and all of its headers'
length and pointers. The message is put
there by grib_seek */
```

c. Initialize variables

```
errmsg[0] = '\0';    /* error buffer */
offset= 0L;          /* byte offset */
decoded_cnt = 0;     /* count of messages */
fout = (FILE *)NULL; /* output file */
grib_data= (float *)NULL; /* decoder will create storage */
```

GRIB USER MANUAL

- d. Set up the name of the file to decode \$GRIB_ENV/data/GRIB0797.tar

```
grib_env_dir= getenv("GRIB_ENV");
if (grib_env_dir==NULL || *grib_env_dir=='\0') {
fprintf(stdout,"Error: Environment variable GRIB_ENV not
defined\n");
exit(0);
}
else sprintf (InFile, "%s/data/GRIB0797.tar",grib_env_dir);
```

- e. Make storage for GRIB header, exit on error

```
if ((nReturn= init_gribhdr (&gh1, errmsg) )) goto bail_out;
```

- f. Create an ASCII⁶ output file, named decoder_ex.output

```
sprintf (OutFn, "decoder_ex.output");
fout = fopen (OutFn, "w");
if (fout == NULL) {
fprintf(stderr,"Failed to open output file %s\n", OutFn);
nReturn = 1; /* error stat */
goto bail_out;
}
```

- g. Loop while there are no errors: Loop until there are no more messages left in the input files.

```
for (offset = 0L; nReturn == 0; offset += gh1->msg_length)
{
```

⁶ American Standard Code for Information Interchange

GRIB USER MANUAL

- h. Call 'grib_seek' to find the next GRIB message in 'infile' starting from the byte 'offset'. If successful, return the entire message and all its information in structure grib_hdr. If it fails, nReturn is non-zero and errmsg contains the error message.

```
if (nReturn= grib_seek(InFile, &offset, Rd_Indexfile, gh1,
errmsg))
{
    fprintf(stdout, "Grib_seek returned non zero stat (%d)\n",
nReturn);
    if (nReturn == 2) break; /* End of file error */
    else goto bail_out;      /* abort if other error */
}
```

- i. Check for warning message from grib_seek; if present, print it out, clear error buffer, and then find next message

```
if (errmsg[0] != '\0')
{ /* NO errors but got a Warning msg from seek */
    fprintf(stdout, "%s; Skip Decoding...\n", errmsg);
    errmsg[0]='\0'; /* reset error to continue */
    gh1->msg_length = 1L; /* set to 1 to bump offset up */
    continue;
}
```

- j. Abort if this message has bad length; if zero length, skip this message, find the next message in the input file

```
if (gh1->msg_length < 0) {
    fprintf(stderr, "Error: message returned had bad length
(%ld)\n", gh1->msg_length);
    goto bail_out;
}
else if (gh1->msg_length == 0) {
    fprintf(stdout, "msg_lenth is Zero, set offset to 1\n");
    gh1->msg_length = 1L; /* set to 1 to bump offset up */
    continue;
}
```

- k. Clear out the GRIB input structures (pds_input, grid_desc_sec, bms_input, bds_head_input)

```
init_dec_struct(&pds, &gds, &bms, &bds_head);
fprintf(stdout, "Decoding message found at %ld bytes
...\n", offset);
```

- l. Decode the message currently in GRIB header; float array must be null, decoder will allocate storage for it. If successful, pds, gds, bds_head, bms, grib_data are returned filled

```
grib_data= (float *) NULL;
if (nReturn = grib_dec ((char *)gh1->entire_msg, &pds,
&gds, &bds_head, &bms, &grib_data, errmsg)) goto bail_out;
```

GRIB USER MANUAL

```
decoded_cnt++;      /* keep count */
```

- m. If the bitmap section (bms) is present in the message then go apply the bitmap to the float array. If successful, the float array is returned expanded to the full #rows by #cols for this grid. The missing data points are filled with the 'fill_value' defined on top

```
if (bms.uslength>0 &&
(nReturn=apply_bitmap(&bms, &grib_data, FILL_VALUE,
bds_head,errmsg)))
goto bail_out;
*****
THE FLOAT DATA ARRAY IS READY TO GO
DATA MAY BE USED IN ANY WAY HERE
PUT CODE FROM HERE TO <<END MARKER>>
```

In this example, the contents of each of the defined header sections of this message along with the first 100 elements in the float data array are displayed on standard output. This is achieved via the call to 'prt_inp_struct'. Then the data array content is appended to the text file named decoder_ex.output, which was previously opened.

```
fprintf(stdout, "\n\t>>> Content of GRIB Msg #d =\n",
decoded_cnt);
prt_inp_struct(&pds, &gds, &bms, &bds_head, &grib_data);

fprintf(fout, "decoded GRIB Msg #d from
$GRIB_ENV/data/GRIB0797.tar\n\n", decoded_cnt);
fprintf(fout, " dtg= %02u%02u%02u%02u Fcstper=%03u\n",
pds.usYear, pds.usMonth, pds.usDay, ds.usHour, pds.usP1);
/* check for NRL's Sub-Parameter Usage */
if (pds.usCenter_sub==99 || pds.usCenter_id==128 || pds.usCenter
_id==129)
&& (pds.usParm_id > 249 && pds.usParm_sub!=999 ))
fprintf(fout, " Parmid=%03u (Sub-Tbl '%c')",
pds.usParm_id-250 + 'A', pds.usParm_sub);
else fprintf(fout, " Parmid=%03u (Main Tbl), ",
pds.usParm_id);
fprintf(fout, "Levelid=%03u, GridId=%03u, lvl1=%05ld\n",
pds.usLevel_id, pds.usGrid_id, pds.usHeight1);
fprintf(fout, " Decimal Scale Factor = %d\n",
pds.sDec_sc_fctr);
```

<p>NOTE: The data are printed up to the precision to which the message was encoded, which is defined by the 'decimal scale factor' in the Product Definition Section.</p>
--

```
D = (int) pds.sDec_sc_fctr;
if (D >= 0)
for (i=0; i<bds_head.ulGrid_size; i=i+5)
```

GRIB USER MANUAL

```
        fprintf(fout,"%5d:  %10.*f  %10.*f  %10.*f  %10.*f
%10.*f\n",
        i+1, D, grib_data[i], D, grib_data[i+1],
        D, grib_data[i+2], D, grib_data[i+3], D,
grib_data[i+4]);
else
    for (i=0; i<bds_head.ulGrid_size; i=i+5)
        fprintf(fout,"%5d:  %10.0f  %10.0f  %10.0f  %10.0f
%10.0f\n", i+1, grib_data[i], grib_data[i+1],
grib_data[i+2], grib_data[i+3], grib_data[i+4]);
<<< END MARKER >>>
STOP SUBSTITUTION HERE
RETAIN THE REMAINING CODE
```

- n. Release the storage allocated for the float array that was reserved with the `malloc` command in the call to `grib_dec`.

```
    if (grib_data!=NULL) { free(grib_data); grib_data = NULL; }
```

- o. Loop again to get the next message in the file.

```
    } /* FOR-loop */
```

- p. This point is reached only if there were no errors. So now change the exit status to no error

```
    nReturn = 0;
```

- q. This section of the code is entered whether the program finished decoding all messages, or if it aborted for some reason. If the program aborted, then the error buffer will contain the cause; print it out;

```
    bail_out:
        if (errmsg[0] != '\0') fprintf(stderr,"\n***ERROR:
%s\n" ,argv[0], errmsg);
        if (grib_data!=NULL) free(grib_data);
        if (fout != NULL) fclose (fout);
```

- r. Release the GRIB header structure.

WARNING: DO NOT OMIT THIS STEP.

```
        free_gribhdr (&gh1);
        exit ( nReturn);
    }
```

To see how to run `decoder_ex` and its results, see **Appendix D**.

GRIB USER MANUAL

5.4.2 PROGRAM 'GETGRIBIEEE.C'

This program shows how to extract and save just the data portion of the GRIB message in an external file. The result is a Binary 32-bit IEEE format file, WITHOUT the 4-byte Header and Trailer. To see how these IEEE files are can be loaded into a C or Fortran program, refer to **Appendix E.2**.

Explanation of `getgribieee`:

- a. Define all of the Include files and global variables:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "dprints.h"          /* for all debug printing */
#include "gribfuncs.h"       /* all GRIB func prototypes */
#define FILL_VALUE -9999.00 /* Value for missing datapts */
char errmsg[2000]; /* leave it big! used by Grib Library */
GRIB_HDR *gh= NULL; /* grib header block, filled by Encoder*/
```

- b. Define local variables in Main function:

```
void main (int argc, char **argv)
{
  int      i;
  char     *grib_env; /* pts to Environment Variable */
  char     InFile[200]; /* current input GRIB filename */
  char     ieee_fn[201]; /* current output Ieee filename */
  int      nReturn=0; /* return status from grib_dec */
  int      S_stat; /* status from grib seek function */
  int      n;
  int      Read_Index=0; /* Zero so Seek will search until Msg*/
  unsigned long msg_length = 0; /* total length of message */
  long     offset; /* offset within the multiple GRIB file,
  indicating end of current msg relative to the top of file*/
  float    *grib_data=NULL; /* array of decoded data values */
  GRIB_HDR *gh1; /* holds info on msg returned from Seek */
  BMS_INPUT bms; /* input structure for bitmap section */
  PDS_INPUT pds; /* product definition section */
  grid_desc_sec gds; /* grid description section */
  BDS_HEAD_INPUT bds_head; /* input structure for bds
  section */
  void      prt_inp_struct ();
  FILE     *fpo, *flist;
```

- c. Clear out name of input file variable

```
InFile[0]='\0';
```

- d. Parse command line arguments, Expecting List filename as the argument

```
if (argc!=2 || (argv[1][0]=='\0' || argv[1][0]=='-')) {
  fprintf(stderr, "Usage: %s List_fn \n",argv[0]);
  exit(0); }
```

GRIB USER MANUAL

e. Open list file for reading, exit if error

```
    if ((flist = fopen(argv[1], "r")) == NULL)
    {    fprintf(stderr,
"Unable to open List file %s;\nAbort Program\n", argv[1]);
    exit(1); }
    fprintf(stdout, "List file is '%s'\n", argv[1]);
```

f. Make storage for GRIB header structure

```
    if (init_gribhdr (&gh1, errmsg))
    {    fprintf(stderr, "%s:  %s\nAbort Program\n", argv[0],
    errmsg);
    fclose(flist); exit(1);
    }
```

g. Loop while not end of List file and no errors exist

```
    while (!ferror(flist) && !feof(flist))
    {
```

(1) Read next entry from Listing file. Entry is name of the next GRIB input file

```
        if ((n=fscanf (flist, "%s", InFile)) != 1) break;
        fprintf(stdout, "\nNext Infile= %s\n", InFile);
```

(2) Keep looping until 'grib_seek' returns error or no more message status

```
        for (offset=0L, errmsg[0]='\0';
        ! (S_stat= grib_seek(InFile, &offset, Read_Index, gh1,
    errmsg));
        offset += (msg_length+ 1L), errmsg[0]='\0')
        {
```

(a) If the message has zero length, loop to call grib_seek again.
Otherwise, do the following:

```
        if ((msg_length = gh1->msg_length) > 0)
        {
            fprintf(stdout, "Decoding message found at %ld
    bytes ... \n",
            offset);
```

(b) Initialize decoder structures

```
        init_dec_struct(&pds, &gds, &bms, &bds_head);
```

(c) Perform GRIB decoding, results in internal PDS, Grid Definition
Section (GDS), Binary Data Section (BDS), Bitmap Section (BMS)
structures and new float data array

```
        grib_data= (float *) NULL;
        if (nReturn = grib_dec((char*)gh1->entire_msg,
    &pds, &gds, &bds_head, &bms, &grib_data, errmsg))
        {
```

GRIB USER MANUAL

```
fprintf(stderr, "%s %s: %s\n", argv[0], InFile, errmsg);
display_gribhdr(gh1);
break;
}
```

(d) Applying bitmap section to data if bitmap section is present

```
if (bms.uslength>0)
if (nReturn=apply_bitmap (&bms, &grib_data, FILL_VALUE,
&bds_head,
errmsg))
{ fprintf(stderr, "%s %s: %s\n", argv[0], InFile,
errmsg);
break;
}
}
```

(e) Create output file to hold the data, named \$InFile.IEEE

Assumption: Only one GRIB message per input file

```
sprintf (ieee_fn, "%s.IEEE",
(strrchr(InFile, '/') ? strrchr(InFile, '/') + 1 :
InFile));

if ( ! (fpo = fopen(ieee_fn, "wb"))) {
    fprintf(stderr, "Cannot open %s for writing\nAbort
Program;\n",
ieee_fn);
fclose(flist);
exit (1);
}

if ( fwrite(grib_data, sizeof(float),
bds_head.ulGrid_size, fpo)
!= bds_head.ulGrid_size)
{
fprintf(stderr,
"Failed to write out %d float elements to %s\nAbort
Program;\n",
bds_head.ulGrid_size, ieee_fn);
fclose(fpo); fclose(flist); exit(1);
}

fprintf(stdout, "Ieee output= %s\n", ieee_fn);
fclose(fpo);

(f) Release storage of the float data array

if (grib_data!=NULL) { free(grib_data); grib_data =
NULL; }

}
```

GRIB USER MANUAL

```
    } /* FOR grib_seek loop*/
```

- (3) If last call to seek GRIB returned 'corrupted len' error, then print as much of the GRIB header structure as possible;

```
    if (S_stat != 0 && errmsg[0] != '\0')
        fprintf(stderr, "%s %s: %s\n", argv[0], InFile, errmsg);
```

- (4) Else, loop to get next entry from List file

```
    }
```

- h. Perform cleaning up and exit

```
    fclose(flist);
    if (grib_data != NULL) { free(grib_data); grib_data = NULL; }
    free_gribhdr (&gh1); /* allocated in ENcode() */
    exit(0);
}
```

To see how this program is executed, refer to **Appendix D.2**.

5.5 GRIB ENCODING

Encoding a gridded data field into the GRIB format is accomplished by filling a defined set of input structures and calling a sequence of library functions. This process is quite simple for a single gridded field loaded into a program. The challenge is in developing consistent mapping from the local data representation scheme to that used in GRIB. To fully understand how GRIB will represent gridded data, one must understand the simple packing scheme employed, the grid systems defined, and the use of enumerated tables of descriptive information explained in the *WMO Manual 306*. **Appendix A** also provides an overview of the structure of a GRIB message.

5.5.1 INTRODUCTION

Before beginning the process of encoding data in GRIB, first verify that GRIB is the proper transfer format for the data. A GRIB message describes one 2-D grid of data represented on a regularly spaced, definable geometry (i.e., a grid system that is computable by parametric equations). Randomly spaced data, such as observation data or a finite element grid, should be sent in Binary Universal Form for Representation of meteorological data (BUFR). 3-D volumes of data can be sent in GRIB (one 2-D grid at a time), assuming that the above grid criteria is met and that the vertical coordinate system is defined by discrete surfaces.

Once the decision has been made to use GRIB, the steps involved in producing GRIB messages are:

- a. Make a list of all fields to be encoded

GRIB USER MANUAL

- b. Map all locally defined parameter, level, geometry, and model names to GRIB code tables, defining new codes as required
- c. Transform the geometry representations to the GRIB scheme
- d. Develop a program that can access all of the required fields from the local data management system one 2-D grid at a time, and can perform any necessary unit conversions and/or geometry transformations as required.
- e. Add the GRIB library encoding structures and function calls to the above program
- f. Select an output file name convention for the GRIB messages

Note that the first three of these are *not* programming steps. It is very easy to quickly develop a GRIB encoding system that goes through the mechanics of creating a GRIB message. However, without the information developed in Steps a-c, there will be no systematic representation of the data in the GRIB messages. Therefore, be sure to spend the time performing Steps a-c prior to developing the GRIB encoding system. The details of performing those important steps are covered in **Section 5.5.2**. Some encoding examples to illustrate Steps d-f are presented in **Section 5.5.3**.

5.5.2 DESCRIBING DATA

The GRIB format treats data as a collection of 2-D gridded fields, each with its own complete set of descriptive information. Therefore, if the local data management system does not treat data in this way, decide how to split data into 2-D grid fields. It is important to determine this now, before proceeding, because all of the discussions and examples to follow assume that the data to be encoded are made up of a collection of 2-D grids.

To completely describe the data set(s), the following information must be encoded for each 2-D gridded data field:

- a. Parameter Name and Units
- b. Level Type and Units
- c. Process used to generate field (Model)
- d. Geometry information
- e. Accuracy of data
- f. Date and Time when field is valid

GRIB USER MANUAL

Once this list is developed, all of the information required to create GRIB tables for the site and fill the input structures required for encoding. The most difficult task in setting up a GRIB encoding program is resolving how to systematically determine all of the above information for each 2-D gridded field as it is processed, and mapping this information to the format of the GRIB input structures.

Part of this process is defining GRIB codes to represent the various parameters, levels, models, and geometries that will be used. The GRIB library includes functions for loading encoding tables that define mappings from local naming conventions to GRIB table codes. For example, suppose a temperature field needs to be encoded, accurate to .01 degrees, which is referenced in the local data set by the term 'air_temp' and is stored in degrees Celsius. Referring to the GRIB Parameter Table (Code Table 2 in the *WMO Manual 306*), the code for temperature data is determined to be '011' and the standard GRIB units are degrees Kelvin. In this case, enter a line in the encoding table as follows:

```
air_temp    0    011    1.0    273.0    3
```

This sets up a parameter mapping that instructs the encoding program to use code '011' in the main table (0) with a scale of 1.0, an offset of 273.0, and a decimal scale factor of 3 when it encounters a field with parameter name = 'air_temp'. Note that the scale and offset included here are not included in the GRIB message, and it is expected that these will be applied to the data array prior to sending the array to the encoding function.

The decimal scale factor is used to scale the data up by powers of 10 to set the desired encoding precision. In this case, a 3 implies that the encoder will scale the data up by 1000 prior to truncating the data to integers, thereby preserving three decimal places of precision in the data. The decimal scale factor should be set to one order of magnitude beyond the desired precision of the data to be encoded.

The above process is repeated in a similar way for all level types, model, and geometry definitions in the local data management system. Level types also include scale and offset values to convert the units defining the levels to standard GRIB units. Refer to **Section 5.6.1.2** for complete details of setting up an encoding table.

After completing the mappings from the local parameter, level type, model, and geometry names to GRIB codes, set up grid definitions for the fields. Although the GRIB standard does not mandate a grid definition section in the message, it is highly recommended because it makes each message completely self-describing. The GRIB grid definition types have been kept internal to the encoding functions, so that the user can always use the same input geometry structure without having to worry about which grid definition type to use. This generic input geometry structure, GEOM_IN, is described in complete detail in **Appendix E.5.7**.

GRIB USER MANUAL

There are a number of ways to handle the inclusion of geometry information in GRIB messages. The most generic way, and consequently the most complicated, is to write a function that loads local geometry information and maps it to the `GEOM_IN` structure as the encoding program processes the data. With this method, the addition of new geometries to the local data management system requires no additional work. However, if only a small number of geometries need be considered, the library supports an external geometry file that stores the required elements of the `GEOM_IN` structure. This can be used to build a database of geometries already in the required format that can be loaded using the `ld_enc_geom` library function. Refer to file `$GRIB_ENV/data/encoder_ex2.geom` for details on the format of the external geometry file format. Encoding Example 2, described in **Section 5.5.4.2**, provides an example using the `ld_enc_geom()` library function.

5.5.3 ENCODING EXAMPLES

It is assumed that the information discussed in **Section 5.5.2** is somehow stored in association with the gridded data fields. It does not matter whether the information is stored in relational database tables or if the information is simply implied from the local file names, as long as the encoding program is able to determine the information for each 2-D grid.

Three example programs are provided to illustrate the basic use of the encoding functions included in the library.

5.5.4 ENCODING CODE EXAMPLES

To create a GRIB message, the user needs to pass to the encoder the array that holds the floating point data and related information stored in the three encoder structures: `GEOM_IN`, `DATA_INPUT`, and `USER_INPUT`. How these items are to be initialized depends upon the user. The following examples illustrate different ways to initialize the required inputs.

5.5.4.1 EXAMPLE 1 - `ENCODER_EX1`

The first example program illustrates the encoding process in its most simplistic form. The three required input structures `GEOM_IN`, `DATA_INPUT` and `USER_INPUT` are manually filled and the input float array is loaded from an IEEE file .

Input file: `$GRIB_ENV/data/IEEE.input`

Output file: `075_237_1997070100012_0011_105.00002.0.grb`

EXPLANATION OF 'encoder_ex1' PROGRAM:

GRIB USER MANUAL

a. Required includes and defines:

```
#include "gribfuncs.h" /* Library function prototypes */
#define INPUT_IEEE_FN "data/IEEE.input" /* to read in */
main ()
{
```

b. Define variables:

```
FILE          *fieee;          /* file pointer to IEEE data file */
GRIB_HDR      *gh= 0;          /* will hold encoded msg & its info
                               initialized to NULL */

DATA_INPUT    data_input;      /* input header struct for Encoder */
GEOM_IN       geom_in;         /* geometry descript for Encoder */
USER_INPUT    user_input;      /* user input for Encoder */
char          *grib_env;       /* defn of Environ var GRIB_ENV */
char          input_fn[200];   /* full path to the ieee file */
char          errmsg[2000];    /* at least 2000 bytes for err msg*/
char          out_fn[50];      /* name of output filename */
float         *flt_arr=NULL;    /* float data array,init to null*/
float         min, max;        /* min and max value of data */
int           cnt,i;           /* working variables */
```

c. Initialize the encoder structures with `init_enc_struct`. This function is of type `Void` and returns nothing.

```
init_enc_struct (&data_input, &geom_in, &user_input);
```

d. Fill in the structures `GEOM_IN`, `DATA_INPUT`, and `USER_INPUT`:

```
strcpy (geom_in.prjn_name, "spherical");
geom_in.nx= 61;          /* num of cols */
geom_in.ny= 51;          /* num of rows */
geom_in.x_int_dis= 22.;  /* X-dir grid length, in meters */
geom_in.y_int_dis= 22.;  /* Y-dir grid length, in meters */
geom_in.parm_1= 0.2;     /* Latitude spacing */
geom_in.parm_2= 0.2;     /* Longitude spacing */
geom_in.parm_3= -1.0;    /* Unused for Spherical */
geom_in.first_lat= 34.;  /* Lat of 1st pt, in degrees */
geom_in.first_lon= 124.; /* Lon of 1st pt, in degrees */
geom_in.last_lat= 0;     /* Lat of last pt, in degrees */
geom_in.last_lon= 0;     /* Lon of last pt, in degrees */
geom_in.scan= 64;        /* pts scan in +i, +j, adjcent pts
                           in i-dir consecutive */

geom_in.usRes_flag= 0;   /* Earth spherical, UV rel. to
                           East/Northerly Dir*/

data_input.usProc_id= 75; /* Model/Generating Process ID
                           (TabA) */
data_input.usGrid_id= 237; /* Grid Identification (Table B)*/
data_input.usParm_id= 11; /* GRIB parameter id */
```

GRIB USER MANUAL

```
data_input.usParm_sub_id= 0; /* GRIB parameter sub-id */
data_input.usLevel_id= 105; /* GRIB level id */
data_input.nLvl_1= 2; /* 1st level value-scaled to an
integer*/
data_input.nLvl_2= 0; /* 2nd level value-scaled to an
integer*/
data_input.nYear= 1995; /* year of data */
data_input.nMonth= 7; /* month of year*/
data_input.nDay= 1; /* day of month */
data_input.nHour= 0; /* hour of day */
data_input.nMinute= 0; /* minute of hour */
data_input.nSecond= 0; /* second of minute */
data_input.usFcst_id=1; /* HOURS Forecast time unit id
Table 4*/
data_input.usFcst_per1= 12; /* forecast time 1 (tau) */
data_input.usFcst_per2= 0; /* forecast time 2 (tau) */
data_input.usTime_range_id= 0; /*Time range indicator-Table 5*/
data_input.usTime_range_avg= 0; /* Number in average */
data_input.usTime_range_mis= 0; /* Number missing from avg */
data_input.nDec_sc_fctr= 1; /* Decimal scale factor */

user_input.chCase_id= '0'; /* User defined Case ID */
user_input.usParm_tbl= 2; /* GRIB Table Version Number*/
user_input.usSub_tbl= 1; /* Local Table Version Number*/
user_input.usCenter_id= 128; /* ID of Orig Center (Table 0) */
user_input.usCenter_sub= 0; /* Sub-Table Entry for Orig Ctr
(Tbl 0)*/
user_input.usTrack_num= 0; /* Tracking ID for data set*/
user_input.usBDS_flag= 0; /* Binary Data Section Flag (Table
11) */
user_input.usGds_bms_id= 128; /* GDS present but not BMS */
user_input.usBit_pack_num= 0; /* No. of bits into which data
is packed. ZERO to encode in Least number of bits */
```

- e. Create a one-dimensional array type float, of dimension specified in the GEOM_IN structure:

```
flt_arr = (float *) malloc(geom_in.nx * geom_in.ny
*sizeof(float));
if (flt_arr == NULL) {
fprintf(stderr, "Failed to allocate storage for Float
array\n");
exit(1);
}
```

- f. Fill Float array with the data for the entire grid point from element 0 to $(\text{geom_in.nx} * \text{geom_in.ny}) - 1$. Data elements should be stored in the order specified by the Scan mode (`Geom_In.scan`). Here, data is loaded from `$GRIB_ENV/data/IEEE.input`.

GRIB USER MANUAL

```
grib_env = getenv ("GRIB_ENV");
if (grib_env == NULL || *grib_env == '\\0') {
fprintf(stderr,"Environment variable GRIB_ENV not
defined\\n");
exit(1);
}

sprintf (input_fn, "%s/%s", grib_env, INPUT_IEEE_FN);
fieee = fopen (input_fn, "rb");
if (fieee == NULL) {
    fprintf(stderr,"Failed to open '%s'\\n", input_fn);
    if (flt_arr != NULL) free(flt_arr);
    exit(1);
}
else {
    if (fread((void*)flt_arr, sizeof(float),
geom_in.nx*geom_in.ny, fieee) != geom_in.nx*geom_in.ny) {
        fprintf(stderr,"Failed to read 'ieee.input'\\n");
        if (flt_arr != NULL) free(flt_arr);
        fclose (fieee);
        exit(1);
    }
}
fclose (fieee);
}
```

- g. Prepare a GRIB_HDR structure via call to `init_gribhdr`. The GRIB Header structure will hold the encoded message and its information. Check the return value of `init_gribhdr`: Zero indicates success, otherwise the cause of failure is in string `errmsg`.

```
if (init_gribhdr (&gh, errmsg) != 0) {
    fprintf(stderr, "%s", errmsg);
    free (flt_arr); /* release storage first */
    exit (1);
}
```

- h. Call `grib_enc` to encode message with the three structure and float array all filled. Upon return, check return status of function `grib_enc`: Zero indicates success, otherwise the cause of failure is in string `errmsg`.

```
if (grib_enc(data_input, user_input, geom_in, flt_arr, gh,
errmsg) != 0) {
    fprintf(stderr,"Abort; error=%s\\n",errmsg);
    if (flt_arr != NULL) free(flt_arr);
    free_gribhdr (&gh);
    exit(1);
}
```

GRIB USER MANUAL

- i. Call function to create the default filename for the encoded message based on the information available in the `DATA_INPUT` and `USER_INPUT` structures. The filename is created and returned in the string `out_fn`. Note that `out_fn` must be long enough to support the created name (50 characters is adequate).

```
make_default_grbfn (data_input, user_input, out_fn);
```

- j. Save the encoded message into an external file with the default name. Check return value of `gribhdr2file`: Zero indicates success, otherwise the cause of failure is in string `errmsg`.

```
if (gribhdr2file (gh, out_fn, errmsg) != 0) {
    fprintf(stderr, "Abort; error= %s\n", errmsg);
    if (flt_arr != NULL) free(flt_arr); /* rel array */
    free_gribhdr (&gh); /* release grib hdr */
    exit(1);
}
```

- k. Before exiting, release storage:

```
if (flt_arr != NULL) free(flt_arr);
free_gribhdr (&gh);
exit(0);
}
```

5.5.4.2 EXAMPLE 2 - ENCODER_EX2

In this example, the three input structures are loaded using the library functions provided, which read the information from external files. Note that these functions MUST be called prior to the call to 'grib_enc'.

```
Input files: $GRIB_ENV/config/encoder.config
             $GRIB_ENV/data/encoder_ex2.geom
             $GRIB_ENV/data/encoder_ex2.info
             $GRIB_ENV/data/IEEE.input
```

```
Output file: 075_237_1997070100012_0011_105.00002.1.grb
```

EXPLANATION OF THE 'encoder_ex2' PROGRAM:

- a. Required Includes and Defines:

```
#include "gribfuncs.h" /* Library function prototypes */
#define CONFIG_FN      "config/encoder.config"
#define GEOM_FN        "data/encoder_ex2.geom"
#define INFO_FN        "data/encoder_ex2.info"
#define IEEE_FN        "data/IEEE.input"
main () {
```

- b. Define variables needed:

```
GRIB_HDR    *gh= 0; /* will hold encoded msg & its info */
DATA_INPUT  data_input; /* input head struct for Encoder */
```

GRIB USER MANUAL

```
GEOM_IN      geom_in;      /* geometry descrip for Encoder */
USER_INPUT   user_input; /* user input from input.dat for Encoder*/
char         errmsg[2000]; /* buffer to hold error message */
char         input_config[200]; /* name of input filename */
char         input_geom[200]; /* name of input filename */
char         input_info[200]; /* name of input filename */
char         input_ieee[200]; /* name of input filename */
char         *grib_env; /* working variable */
char         out_fn[50]; /* name of output filename */
float        *flt_arr=NULL; /* float data array, init null*/
float        min, max; /* min and max value of data */
int          cnt, i; /* working variables */
```

- c. Get the Environment Variable GRIB_ENV first:

```
grib_env = getenv ("GRIB_ENV");
if (grib_env == NULL || *grib_env == '\0') {
    fprintf(stderr, "Environment variable GRIB_ENV not
defined\n");
    exit(1);
}
```

- d. Clear out the 3 encoder structures:

```
init_enc_struct (&data_input, &geom_in, &user_input);
```

- e. Build the names for the 3 input files to load:

```
sprintf (input_config, "%s/%s", grib_env, (char *)CONFIG_FN);
sprintf (input_geom, "%s/%s", grib_env, (char *)GEOM_FN);
sprintf (input_info, "%s/%s", grib_env, (char *)INFO_FN);
sprintf (input_ieee, "%s/%s", grib_env, (char *)IEEE_FN);
```

- f. Call the three library functions to load these three files into the three encoder structures GEOM_IN, DATA_INPUT, USER_INPUT. Check each function's return value for error: zero indicates success, else error message is stored in errmsg:

```
if (ld_enc_geomfile (input_geom, &geom_in, errmsg) != 0 ||
ld_enc_ffinfo (input_info, model_id, &data_input, errmsg) != 0 ||
ld_enc_config (input_config, &user_input, errmsg) != 0 )
{
    fprintf(stderr, "Fatal error= %s\n", errmsg); exit(1);
}
```

- g. Make storage for a one-dimensional array of type Float. The array needs to be large enough to support the grid size shows in GEOM_IN:

```
if (!(flt_arr = (float *)malloc(geom_in.nx*geom_in.ny
*sizeof(float)))){
    fprintf(stderr, "Failed to malloc Float array\n");
    exit(1);
}
```

- h. Now call library function to load the IEEE file into float array. Again check return value from function, non-zero means error occurred:

```
if (ld_enc_ieeeff (input_ieee, flt_arr, geom_in.nx*geom_in.ny,
errmsg)){
```

GRIB USER MANUAL

```
    fprintf(stderr, "Failed to load IEEE file, errmsg=%s\n",
errmsg);
    exit(1);
}
```

- i. Allocate storage for GRIB_HDR structure. Check return status for error:

```
if (init_gribhdr (&gh, errmsg)) {
    fprintf(stderr, "Abort; error=%s\n", errmsg);
    if (flt_arr != NULL) free(flt_arr);
    exit(1);
}
```

- j. Call encoder routine passing it the 3 structures and the float array. If successful, the encoded message is returned in the GRIB_HDR. Check return status for non-zero status:

```
if (grib_enc(data_input, user_input, geom_in, flt_arr, gh,
    errmsg) != 0)
{
    fprintf(stderr, "Abort; error=%s\n", errmsg);
    if (flt_arr != NULL) free(flt_arr);
    free_gribhdr (&gh);
    exit(1);
}
```

- k. Form the default name for GRIB file using information available in the structures DATA_INPUT and USER_INPUT. The name is returned in string out_fn:

```
make_default_grbfn (data_input, user_input, out_fn);
```

- l. Save the encoded message in GRIB_HDR out to external file under the default GRIB filename. Check return status for non-zero status:

```
if ( gribhdr2file (gh, out_fn, errmsg) != 0)
{
    fprintf(stderr, "Abort; error= %s\n", errmsg);
    if (flt_arr != NULL) free(flt_arr);
    free_gribhdr (&gh);
    exit(1);
}
```

- m. Free up storage used before exiting:

```
if (flt_arr != NULL) free(flt_arr);
free_gribhdr (&gh);
exit(0);
}
```

GRIB USER MANUAL

5.5.4.3 EXAMPLE 3 - ENCODER_EX3

This third example shows how to encode a list of GRIB messages from the user-provided Geometry, Model, and list file containing names of the IEEE files. Each encoded message will be saved to an external file using the default GRIB file name provided by the library. This example program represents a realistic GRIB encoding system in which multiple geometry definitions can be stored in a flat-file database and referenced by name, and an external encoding table is used to maintain local definitions of parameter and level types, models, and geometries. This program represents an early version of a GRIB encoder used at the NRL, Monterey MEL Resource site.

RUNNING 'encoder_ex3'

This program expects three command line arguments: Model type, Geometry name, and List filename.

```
encoder_ex3 NORAPS2 ptmugu_61x51
$GRIB_ENV/data/encoder_ex3.list
```

Input files:

- `$GRIB_ENV/config/encoder.config:`
File holding the configuration information for the encoder.
- `$GRIB_ENV/data/'$GEOM'.geom`
Geometry information file. Since the command line specifies geometry `ptmugu_61x51`, program will automatically look for the file `'$GRIB_ENV/data/ptmugu_61x51.geom'`
- `$GRIB_ENV/data/encoder_ex3.list`
This file contains names of the input IEEE files to be used for encoding. Each entry appears on a single line and may not contain a path. The program automatically tacks the path `$GRIB_ENV/data/` in front of each file name. Here is the listing of `'$GRIB_ENV/data/encoder_ex3.list'`

```
ieee.pres.msl.2.0.1997070100.000
ieee.pres.msl.2.0.1997070100.012
ieee.wnd_ucmp.ht_sfc.10.0.1997070100.000
ieee.wnd_ucmp.ht_sfc.10.0.1997070100.012
```
- IEEE input files
These files hold the IEEE data used for each message to encode. The filename format is:
`"ieee."parname.lv1type.lv11.lv12.yyyymmddhh.tau`
as listed in the list file. All input files must reside in `$GRIB_ENV/data/`.

Output files:

GRIB USER MANUAL

From the four input "ieee." entries in the Listing files, four GRIB messages are encoded and they are stored in separate files =

```
016_054_1997070100000_0001_102.00002.3.grb
016_054_1997070100000_0033_105.00010.3.grb
016_054_1997070100012_0001_102.00002.3.grb
016_054_1997070100012_0033_105.00010.3.grb
```

EXPLANATION OF THE 'encoder_ex3' PROGRAM:

a. Required includes and defines:

```
#include <stdio.h>
#include <stdlib.h>
#include "gribfuncs.h"      /* Library function prototypes */
#include "grib_lookup.h"   /* Conversion Info */
#include "dprints.h"       /* Debug printing info*/

#define CONFIG_FN          "config/encoder.config"

extern PARM_DEFN db_parm_tbl[]; /* holds parm conversion info */
extern LVL_DEFN db_lvl_tbl[];  /* holds level conversion info */
extern MODEL_DEFN db mdl_tbl[]; /* holds model conversion info */
extern GEOM_DEFN db_geom_tbl[]; /* holds Geom conversion info */
```

b. Program source:

```
main (int argc, char *argv[])
{
(1) Define the local variables.

FILE      *flist = NULL; /* List file pointer */
GRIB_HDR  *gh= 0;      /* will hold encoded msg & its info*/
DATA_INPUT data_input; /* Encoder input header structure */
GEOM_IN   geom_in;    /* geometry description for Encoder*/
USER_INPUT user_input; /* Encoder user input from input.dat */
char      *geom_name; /* cmd line input */
char      *model_type; /* cmd line input */
char      *list_fn;   /* name of List filename */
char      *grib_env;  /* working variable */
char      line[100],temp[100],dummy[100]; /* working vars */
char      *p1,*p2;
char      errmsg[2000]; /* buffer to hold error message */
char      config_fn[200]; /* name of input filename */
char      lookup_fn[100]; /* name of Conversion file */
char      geom_fn[100];   /* name of Geometry file */
char      ieee_fn[200];  /* name of input filename */
char      out_fn[50];    /* name of output filename */
float     *flt_arr=NULL; /* array to hold float data, init
                        to null */
float     scl,ref,min, max; /* min and max value of data */
int       id,cnt,i;      /* working variables */
int       quit;         /* set to quit */
```

GRIB USER MANUAL

```
int      stat;                /* working var */
unsigned long uldtg;
```

- (2) Parse the Command Line Arguments, expecting 3.

```
if (argc != 4) {
fprintf(stderr, "Usage= %s Modeltype Geomname
  List_fn\n\n", argv[0]);
exit(1);
}
fprintf(stdout, "\nStarting %s\n", argv[0]);
model_type = argv[1]; geom_name= argv[2]; list_fn=
  argv[3];
```

- (3) Check Environment Var 'GRIB_ENV', exit if not available.

```
grib_env = getenv ("GRIB_ENV");
if (grib_env == NULL || *grib_env == '\0') {
fprintf(stderr, "Environment variable GRIB_ENV not
defined\n");
  exit(1);
}
fprintf(stdout, "GRIB_ENV = %s\n", grib_env);
```

- (4) Clear out the three internal Encoder structures.

```
fprintf(stdout, "Clear out the 3 Encoder structures \n");
init_enc_struct (&data_input, &geom_in, &user_input);
```

- (5) Load the Encoder Configuration file.

```
sprintf (config_fn, "%s/%s", grib_env, (char
*)CONFIG_FN);
fprintf(stdout, "Loading Config file= %s\n", config_fn);
if (ld_enc_config (config_fn, &user_input, errmsg) != 0 )
{
fprintf(stderr, "Fatal error= %s\n", errmsg);
exit(1);
}
```

- (6) Build Lookup filename then load it. This file contains the information needed on Parameters, Models, Geometries, and Levels.

```
sprintf (lookup_fn, "%s/tables/neons2grib.%d.%d",
grib_env,
user_input.usParm_tbl, user_input.usSub_tbl);
fprintf(stdout, "Loading Lookup file= %s\n", lookup_fn);

if (ld_enc_lookup (lookup_fn, errmsg)) {
fprintf(stderr, "Fatal error= %s\n", errmsg);
exit(1);
}
```

GRIB USER MANUAL

- (7) Check for the input Geometry provided by the user. Quit if it does not match any of the geometries loaded from the Table file; else note the ID of the matched Geometry Name.

```
for (id=0; id < NGEOM; id++) if (!strcmp
(db_geom_tbl[id].db_name, geom_name))
{ data_input.usGrid_id = id; break; }
/* found it */
if (id==NGEOM) {
fprintf(stderr,"Error: invalid Geom %s\n", geom_name);
exit(1);
}
```

- (8) Check for the input Model provided by the user. Quit if it does not match any of the models loaded from the Table file; else note the ID of the matched Model Type.

```
for (id=0; id < NMODEL; id++)
if (!strcmp (db_mdl_tbl[id].db_name, model_type))
{ data_input.usProc_id = id; break; }
/* found it */
if (id==NMODEL) {
fprintf(stderr,"Error: invalid Model %s\n", model_type);
exit(1);
}
```

- (9) Build the name of ".geom" filename based on the input Geometry Name.

```
sprintf (geom_fn, "%s/data/%s.geom", grib_env,
geom_name);
fprintf(stdout,"call ld_enc_geomfile (%s)\n", geom_fn);
```

- (10) Load the Geometry file into the internal GEOM_IN struct

```
if (ld_enc_geomfile (geom_fn, &geom_in, errmsg) != 0 ) {
fprintf(stderr,"Fatal error= %s\n", errmsg);
exit(1);
}
```

- (11) Create storage for an array to hold the Data for this grid type, with dimensions based on the number of columns and rows for this geometry. Quit if procedure fails.

```
fprintf(stdout,"Malloc float array %d by %d\n",
geom_in.nx, geom_in.ny);
if (!(flt_arr =
(float *) malloc(geom_in.nx * geom_in.ny
*sizeof(float))) {
fprintf(stderr,"Failed to malloc Float array\n");
exit(1);
}
```

GRIB USER MANUAL

- (12) Open the List file for reading, quit on error.

```
fprintf(stdout,"Prepare List file '%s' for reading\n",
list_fn);

if ( !(flist = fopen (list_fn, "r")) ) {
    fprintf(stderr,"Unable to open Listfile %s\n",
list_fn);
    if (flt_arr != NULL) free(flt_arr); /* release float
array */
    exit(1);
}
```

- (13) Allocate storage and initialize GRIB_HDR structure, quit on error.

```
if (init_gribhdr (&gh, errmsg)) {
    fprintf(stderr,"Abort; error=%s\n", errmsg);
    if (flist) fclose (flist);
    if (flt_arr != NULL) free(flt_arr); /* release float
array */
    exit(1);
}
```

- (14) Loop: !create a GRIB msg per entry in List file

```
for (stat=0; !feof(flist) && !ferror(flist); )
{
(a) Read next entry from List file
    memset ((void*)line, '\0', sizeof line);
    memset ((void*)dummy, '\0', sizeof dummy);
    memset ((void*)temp, '\0', sizeof temp);
    if (fgets(line, 100, flist) == NULL) break;
    if (sscanf (line, "%s%s", temp, dummy) != 1) {
        fprintf(stderr,"Invalid List_fn entry: [%s %s...]\n",
temp,dummy);
        continue; }

```

```
    fprintf(stdout,"\n[ %s ]=\n", temp);
```

- (b) Extract out the field information from the entry such as parmname, lvtype, lvl1, lvl2, dtg, forecast period. Entry has format=
'ieee.parmname.lvtype.l1.l2.yyyymmddhh.tau'

```
for (p1=temp, quit=i=0; !quit && i < 7 ; i++, p1=p2+1)
{
    if ((p2=(char *)strchr(p1, '.'))==NULL)
        p2=temp+strlen(temp);
    /* Extract */
    strncpy (dummy, p1, p2-p1);    dummy [ p2-p1] = '\0';

```

GRIB USER MANUAL

```
switch (i) {
  case 0: /* 'ieee.' part */
    if (strcmp (dummy, "ieee")) {
      fprintf(stderr, "No 'ieee.', drop [%s]\n", temp);
      quit=1; }
    break;

    case 1: /* PARM part,
fill data_input's usParmid, usParmsub, nDec_sc_fctr
*/
    if (map_parm (dummy, &data_input, &scl, &ref,
errmsg)) {
      fprintf(stderr, "%s, drop [%s]\n", errmsg, temp);
      quit=1; }
    break;

    case 2: /* LEVEL part,
go fill data_INPUT'S usLevelid, SCALE up lvl1 & lvl2
which currently have not been read in yet */
    if (map_lvl (dummy, &data_input, &scl, &ref, errmsg))
    {
      fprintf(stderr, "%s, drop [%s]\n", errmsg, temp);
      quit=1; }
    break;

    case 3: /* LEVEL_1 part */
    if (strcspn (dummy, "0123456789") != 0) {
      fprintf(stderr, "Bad level_1, drop [%s]\n", temp);
      quit=1; }
    else data_input.nLvl_1 = scl*atoi(dummy) + ref;
    break;

    case 4: /* LEVEL_2 part */
    if (strcspn (dummy, "0123456789") != 0) {
      fprintf(stderr, "Bad level_2, drop [%s]\n", temp);
      quit=1; }
    else data_input.nLvl_2 = scl*atoi(dummy) + ref;
    break;

    case 5: /* DTG part */
    if (strlen(dummy)!=10 ||
        strcspn (dummy, "0123456789")!= 0)
      { fprintf(stderr, "Bad DTG, drop [%s]\n", temp);
        quit=1; }
    else {
      uldtg = (unsigned long)atol (dummy);
      data_input.nYear = uldtg / 1000000;
      data_input.nMonth = (uldtg / 10000) % 100;
      data_input.nDay = (uldtg / 100) % 100;
      data_input.nHour = (uldtg % 100);
      if (*(p2+1) == '\\0') { fprintf(stderr,
"Missing Forecast Period, drop [%s]\n", temp);
        quit=1; }
      }
    break;

    case 6:
```

GRIB USER MANUAL

```
/* Forecast Period part */
if (strcspn (dummy, "0123456789") != 0) {
    fprintf(stderr, "Bad Forecast Period, drop [%s]\n",
        temp); quit=1; }
else
{
    data_input.usFcst_per1 = (unsigned
    short)atoi(dummy);
    if (*(p2+1) != '\0')
        { fprintf(stderr,
            "Invalid info AFTER Fcst_per, drop [%s]\n",
            temp); quit=1;
        }
    }
break;
} /* Switch */
} /* For */
```

- (c) Check if the extraction of info was successful. If not then skip this entry, loop again to get the next entry in List file;

```
if (quit) continue;
if (i < 6) {
    fprintf(stderr,
    "Proper entry=
    'ieee.$parm.$lv1.$lv11.$lv12.yyyyymmddhh.tau', "\
    "drop [%s]\n", temp);
    continue;
}
```

- (d) Go load IEEE file into Float array, skip if failed.

```
sprintf (ieee_fn, "%s/data/%s", grib_env, temp);
fprintf(stdout, "call ld_enc_ieeeff (%s)\n", ieee_fn);
if (ld_enc_ieeeff (ieee_fn, flt_arr, geom_in.nx *
    geom_in.ny, errmsg)) {
    fprintf(stderr, "Failed to load IEEE file %s,
    errmsg=%s\n",
    ieee_fn, errmsg);
    continue;
}
```

- (e) If desired, find the Range of Data and count #Zeros.

```
cnt = 0;
min = max = flt_arr[0];
for (i=geom_in.nx*geom_in.ny-1; i > 0; i--) {
    if (min > flt_arr[i]) min = flt_arr[i];
    if (max < flt_arr[i]) max = flt_arr[i];
    if (flt_arr[i] == 0.0) cnt++;
}
fprintf(stdout, "--> MIN value= %lf, MAX= %lf, %ld
    Zeros\n",
    min, max, cnt);
```

GRIB USER MANUAL

- (f) Change the Case ID character to flag this as example #3. This character is the 'c' part of the default filename provided by the GRIB library with the format.

```
'MID_GID_yyyymmddhhhtt_PID_LID_lv11.c.grb'  
user_input.chCase_id='3';
```

- (g) Go Encode the message now, quit on error.

```
if (grib_enc (data_input, user_input, geom_in, flt_arr,  
gh,  
errmsg)!= 0) {  
fprintf(stderr, "Abort; error=%s\n", errmsg);  
stat=1; break;  
}
```

- (h) Form a filename for the output file that reflects the type of message.
The format of the filename is:

```
'MID_GID_yyyymmddhhhtt_PIndx_LID_lv11.c.grb'  
make_default_grbfn (data_input, user_input, out_fn);
```

- (i) Write the encoded message out to external file using the default name.

```
if ( gribhdr2file (gh, out_fn, errmsg) != 0) {  
fprintf(stderr, "Abort; error= %s\n", errmsg);  
stat=1; break;  
}  
} /* loop for each Ieee fn in List_fn */
```

- (15) Close List file, free storage of data array.

```
if (flist != NULL) fclose (flist);  
if (flt_arr != NULL) free(flt_arr);
```

- (16) Free up storage used by GRIB Header,

CAUTION: DON'T FORGET THIS STEP
--

```
free_gribhdr (&gh);
```

- (17) Exit the program.

```
exit(stat);  
}
```

5.6 GRIB TABLE MANAGEMENT

This section is primarily intended for users who intend to generate GRIB data. Developers of custom decoding applications may need to reference the Decoder Table description in Section 5.6.1.1, but it should be stressed that these users only need to access the decoder table - their application should not modify the decoder table in any way, as this is maintained by the GRIB encoding site (originating center) and/or MEL.

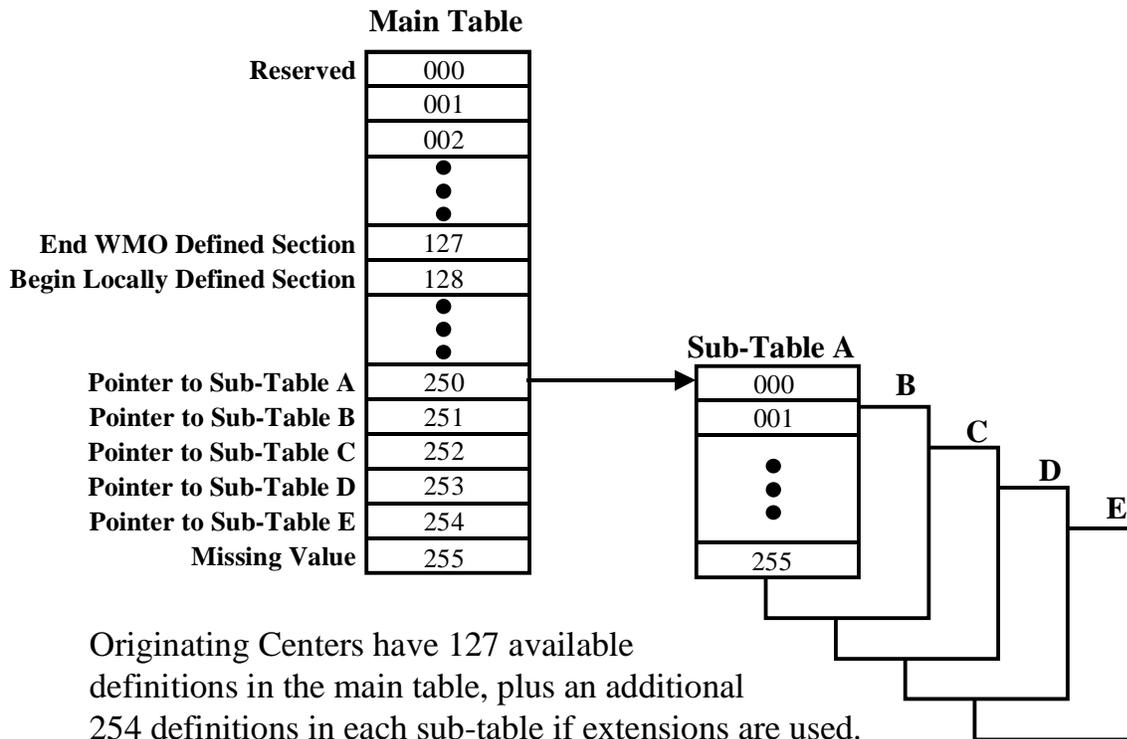
To allow for uniqueness among multiple originating centers' GRIB definitions, a table format has been developed as part of the MEL GRIB Software Library. This table format is not supported by the WMO GRIB Standard, and was developed for use primarily by MEL Resource Sites only, though it could be used by any GRIB producing site. The use of external tables does not violate the GRIB standard in any way; it simply provides a mechanism by which a single application can process GRIB data from multiple sites without any internal code changes.

There are four binary codes in the Product Definition Section that refer to a table of enumerated definitions. They are the Generating Process ID, the Grid Definition ID, the Parameter ID, and the Level Type ID. In each case, the ID must fit in one octet (or byte), which means there are 256 possible values, from 0 to 255. In the case of the Generating Process ID and Grid Definition ID, all definitions are left up to the originating center; the WMO has not defined any standard set of grids or generating processes. In the case of the Level type ID, there is a standard set of definitions by the WMO, but there is also room in the table for some additions by the originating center.

The parameter ID table contains the WMO definitions confined to the first 128 codes, while the second 128 codes are open for definition by the originating center. The MEL GRIB software has implemented a sub-table extension for the parameter ID table where the codes 250 through 254 each indicate (or point to) a secondary table of another 255 possible definitions (see Figure 3). The structure of the sub-table definitions is described in the table format discussion below. The sub-table information is included in a GRIB message by storing the sub-table indicator (code 250 through 254) in the parameter ID position of the PDS (octet 9), and storing the code ID defined within the sub-table in sub-parameter ID of the PDS (octet 45). The sub-parameter ID stored in octet 45 is an extension to the GRIB standard, and is fully described in Appendix C. Because only the MEL GRIB library functions know how to access this extension, users are encouraged to use the 128 codes available in the main table before defining parameters in the sub-tables.

GRIB USER MANUAL

Indicator of Parameter with MEL Extension for Sub-Tables



Originating Centers have 127 available definitions in the main table, plus an additional 254 definitions in each sub-table if extensions are used.

Figure 3. WMO Code Table 2

There are two table files related to code definitions that must be maintained by any center producing GRIB messages. The first is the encoding table, which provides a mapping from the local reference names for parameters, levels, geometries, and models to GRIB codes. The encoding table also includes some required encoding information for each parameter and level type. This table is meant for local use at the encoding site only. The second table is the decoding table, which provides a mapping from the GRIB codes decoding programs will encounter in processing messages to descriptive information about gridded data fields contained in the messages. This table is intended for distribution to the GRIB users, and is to be made available on the MEL GRIB Table FTP site (see **Section 5.3.2**).

5.6.1 EXTERNAL TABLE FORMAT

The information required to decode and encode a GRIB message is stored in two separate files. In this document, they will be referred to as the Decoder and Encoder Table files. The MEL GRIB Software Library load functions require these files to be in the format described below.

The following rules need to be followed when creating or altering the Decoder and Encoder Table files:

1. Tab characters are not allowed.

GRIB USER MANUAL

2. Lines that start with '#' are considered to be comments and are skipped.
3. Each table file is made up of multiple tables. Each table within a file must begin with the proper Header Section and must be followed by all of its entries. The load function will assume it is at the end of the current table when it reads the next table's Header Section.
4. Lines that do not have as many arguments as expected will be dropped by the load function. A warning message will be displayed.
5. Table entries (a single line in the file) must have at least two spaces between each attribute, unless specified otherwise below.

5.6.1.1 THE DECODER TABLE FILE

The information needed to decode a GRIB message's Parameter ID, Level Type ID, Model ID and Geometry ID is organized into separate tables in the Decoder Table file. These tables *must* appear in the following order: the Main Parameter table, the Parameter Sub-Tables A through E, the Level Type table, the Model Table, and the Geometry Table.

GRIB Table 2: Parameter Definitions

The GRIB Parameter table provides a mapping from GRIB parameter codes to text definitions of parameter and unit. The WMO provides a default set of parameter definitions in Code Table 2 of Reference C. These WMO definitions are confined to the range 1 to 127 in the Main Parameter table. Additional Parameters may be defined for local use in the range 128 - 255 of the main table, and in the range 1 to 255 of the five sub-tables (A through E). The use of sub-tables allows for an additional 1275 parameter definitions, in addition to the 255 definitions in the main table. The following rules must be followed to ensure proper Parameter Table loading:

1. Header Section:

```
GRIB Table 2
Code Figure      Field Parameter      Unit
=====
002              Pressure reduced to MSL    Pa
003              Pressure tendency         Pa/s
```

The first three lines above make up the Header Section of the Main Parameter table in the Decoder file. They all start at the first column of each line. The Sub-Table Header Section shall read "GRIB Table 2 - Sub A" through "GRIB Table 2 - Sub E" rather than "GRIB Table 2". Note that these Header Sections **MUST** be present whether local tables are used or not.

```
#####
GRIB Table 2 - Sub A
Code Figure Field Parameter Unit
=====
```

GRIB USER MANUAL

```
#####  
GRIB Table 2 - Sub B  
Code Figure Field Parameter Unit  
===== =====  
#####  
GRIB Table 2 - Sub C  
Code Figure Field Parameter Unit  
===== =====  
#####  
GRIB Table 2 - Sub D  
Code Figure Field Parameter Unit  
===== =====  
#####  
GRIB Table 2 - Sub E  
Code Figure Field Parameter Unit  
===== =====
```

2. The GRIB code in the Main Table and all of the Sub-Tables must fall between 001 and 255 (GRIB code 000 is reserved and not used). Each GRIB code is only accepted once by the Load function, so duplicate GRIB code definitions are dropped, even if the Description and Unit are different.

3. The Description field can contain one or more words (upper case or lower case) separated by a space. It cannot exceed 74 characters in length and must be followed by at least 2 spaces before the Unit field begins. Uniqueness of this field is not checked.

4. The Unit field can contain one or more words (upper case or lower case) separated by a space. It cannot exceed 24 characters in length and defaults to '-' where the Unit is not applicable. Uniqueness of this field is not checked.

Lines with invalid GRIB codes, missing Description or Unit will be dropped by the load function and a warning message will be displayed.

GRIB Table 3: Level Type Definitions

The following rules must be followed to ensure proper Level Table loading:

1. The Level Header Section contains 4 lines and all start at column 1. This is the only table that does not have a line with '=== ' as the last Header line.

```
GRIB Table 3: Level Definitions  
Line 1: Level ID | Number of Octets | Meaning  
Line 2: Contents of octet 11 (optional)  
Line 3: Contents of octet 12 (optional)
```

GRIB USER MANUAL

Each record of the level table has up to three lines depending on the value of "Number of Octets" field. The first line provides three pieces of information: a GRIB code, a count of Octets and the Meaning (description) of the Level. All three of these items must be separated with at least two spaces (Vertical bar '|' is only for clarification and is not needed). Lines 2 and 3 are optional and their existence is dictated by the Number of Octets, see below.

2. The GRIB code must fall between 001 and 255 (GRIB code 000 is reserved and not used). Each GRIB code is only accepted once by the Load function, so duplicate GRIB code definitions are dropped.

3. The Number of Octets indicates how many octets are used to define each surface of the level type. Valid values are 0, 1, or 2.

"0" means that octets 11 and 12 are not defined, and that there will be no more lines to follow.

"1" indicates a layer with octet 11 defining the top surface and octet 12 defining the bottom surface. A value of "1" therefore implies that there will be two more lines in the table.

"2" indicates a level defined by one 2-byte value stored in both octets 11 and 12. There will be one more line in the table in this case.

4. Meaning is a description of the Level. It may contain one or more upper or lower case words separated by a space, and cannot exceed 99 characters. Uniqueness of this field is not checked.

5. The Octet 11 content line is only present if number of Octets is 1 or 2. It may contain one or more upper or lower case words separated by a space, and cannot exceed 99 characters. Uniqueness of this field is not checked.

6. The Octet 12 content line is only present if number of Octets is 1. It may contain one or more upper or lower case words separated by a space, and cannot exceed 99 characters. Uniqueness of this field is not checked.

Some examples follow:

002 0 Cloud base level

003 0 Level of cloud tops

100 2 Isobaric surface

Pressure in hPa

101 1 Layer between two isobaric surfaces

Pressure of top in kPa

Pressure of bottom in kPa

103 2 Specified altitude above mean sea level

Altitude in meters

104 1 Layer between two specified altitudes above mean sea level

GRIB USER MANUAL

Altitude of top in hm
Altitude of bottom in hm
105 2 Specified height level above ground
Height in meters

Lines with invalid GRIB codes, a missing description, or an inconsistency between the number of octets field and the number of descriptive lines that follow will be dropped by the load function and a warning message will be displayed.

GRIB Model Definitions

The following rules must be followed to ensure proper Model Table loading:

1. The Model Header Section contains 3 lines and all start at column 1.

```
GRIB Table - Generating Process Defs (Octet 6 of PDS)
Code Figure      Model Name
=====          =====
```

2. Each record of the table contains a GRIB code and a description of the model or process that generated the field. The GRIB code must fall between 001 and 255 (GRIB code 000 is reserved and not used). Duplicate codes are dropped.

3. The Model Name may be one or more words separated by a space and cannot exceed 60 characters.

Lines with invalid GRIB codes or a missing description will be dropped by the load function and a warning message will be displayed.

GRIB Geometry Definitions

The following rules must be followed to ensure proper Geometry Table loading:

1. The Geometry Header Section contains 3 lines and all start at column 1.

```
GRIB Table - Pre-defined geometries (Octet 7 of PDS)
Code Figure      Geometry Name
=====          =====
```

2. Each record of the table contains a GRIB code and a description of the geometry. The GRIB code must fall between 001 and 255 (GRIB code 000 is reserved and not used). Duplicate codes are dropped.

3. The Geometry Name may be one or more words separated by a space, but cannot exceed 60 characters.

<p>NOTE: Lines with invalid GRIB codes or a missing description will be dropped by the load function and a warning message will be displayed.</p>
--

GRIB USER MANUAL

5.6.1.2 THE ENCODER TABLE FILE

The information needed to encode the Parameter, Level Type, Model and Geometry associated with a gridded data field is organized into separate tables in the Encoder Table file. These tables *must* appear in the following order: the Parameter Tables, Level Table, Model Table, and Geometry Table.

The Parameter Table

The Parameter Table is made up of a main table and five sub-tables labeled A through E. Definitions in the sub-tables are encoded by placing the sub-table identifier in the parameter identification octet of the PDS, and placing the sub-table definition in octet 45 of the PDS. The sub-table identifiers are referenced as shown in Table 2.

Table 2. Sub-Table Identifiers

<i>Sub-Table</i>	<i>Identifier</i>
A	250
B	251
C	252
D	253
E	254

The Parameter and Units provided by the WMO are confined to the range 1 to 127 in the Main Parameter Table. Additional Parameters may be defined for local use and are stored in the range 128 - 255 of the main table, and in the five sub-tables (A through E). The use of sub-tables allows for an additional 1275 parameter definitions, in addition to the 255 definitions in the main table. Each record of the Parameter table contains six pieces of information: a Parameter Name, a Table Code, a GRIB code, a Scale Factor, an Offset, and a Decimal Scale Factor.

The following format must be followed to ensure proper Parameter Table loading:

1. Header Section:

```
XXXXX to GRIB Parameter Table
XXXXX FIELD      TABLE CODE  GRIB CODE      SCALE      OFFSET      DSF
=====
wnd_ucmp          0           033           1.000000   0.000000   5
wnd_vert_vel      0           039           100.000000 0.000000   5
```

GRIB USER MANUAL

mn_dpth	0	147	1.000000	0.000000	1
cloud_coverage	A	001	1.000000	0.000000	1

The first three lines above make up the Header Section of the Parameter table in the Encoder file. They all start at the first column of the line. The XXXXX is meant to refer to the name of the local data management system.

2. The Field name consists of one contiguous name, not to exceed 30 characters, to indicate how the parameter is referenced in the local dbms.
3. The Table Code is either 0, A, B, C, D, or E. Codes A through E indicate at the GRIB code that follows is part of the specified Sub-Table. Code 0, implies a definition in the main table.
4. The GRIB code in the Main Table and any of the Sub-Tables must fall between 001 and 255 (GRIB code 000 is reserved and should not be used). Each GRIB code is only accepted once by the Load function so duplicate GRIB code definitions are dropped even if the Description and Unit are different.
5. The Scale and Offset are used to convert the parameter's unit to GRIB standard units.
6. The Decimal Scale Factor is an integer that is used to scale the data up by powers of 10 to set the desired encoding precision. A value of 3 implies that the data will be scaled up by 1000 prior to truncating the data to integers, thereby preserving precision in the data to 3 decimal places. It is recommended that the decimal scale factor be set to one order of magnitude beyond the precision desired.

Lines with missing arguments are dropped and a warning message is displayed.

The Level Table

The following format must be followed to ensure proper Level Table loading:

1. The Header Section:

```
XXXXX to GRIB Level Table
XXXXX Level Type      GRIB CODE      SCALE      OFFSET
=====
surface                001            1.000000    0.000000
atms_lay               104            0.010000    0.000000
sgma_lvl              107           10000.000000  0.000000
isnt_lvl              113            0.000000    273.160004
```

The first three lines above make up the Header Section of the Level table in the Encoder file. They all start at the first column of the line.

GRIB USER MANUAL

2. The Level Type consists of one contiguous name, not to exceed 30 characters, to indicate that the level is referenced in the local dbms.
3. The GRIB Code should fall in the range of 1 through 255. GRIB Code 000 is reserved and not used. Duplicate Code definitions are dropped.
4. The Scale and Offset are used to convert the level's unit to GRIB standard units.

The Model Table

The following format must be followed to ensure proper Model Table loading:

1. The Header Section:

```
XXXXX to GRIB Model Table
XXXXX Model Name          GRIB CODE
=====                   =====
NORAPS                     001
COAMPS                     002
NOGAPS                     003
```

The first three lines above make up the Header Section of the Model table in the Encoder file. They all start at the first column of the line.

2. The Model Name consists of one contiguous name, not to exceed 30 characters, used to describe the Model.
3. The GRIB Code should fall in the range of 1 through 255. GRIB Code 000 is reserved and not used. Duplicate Code definitions are dropped.

The Geometry Table

The following format must be followed to ensure proper Geometry Table loading:

1. The Header Section:

```
XXXXX to GRIB Geometry Table
XXXXX Geometry Name      GRIB CODE
=====                   =====
mediterranean_109x82    001
persian_gulf_NORAPS_63x63 002
global_144x288          003
```

The first three lines above make up the Header Section of the Geometry table in the Encoder file. They all start at the first column of the line.

2. The Geometry Name consists of one contiguous name, not to exceed 30 characters, to describe the Geometry.

GRIB USER MANUAL

3. The GRIB Code should fall in the range of 1 through 255. GRIB Code 000 is reserved and not used. Duplicate Code definitions are dropped.

5.7 FUNCTION DEFINITIONS

NOTE: In any of the functions discussed below, please use this key. ARGUMENTS
(*Input*>> = input, <<*Output* = output, <<*Input and Output*>> = (input and output)

5.7.1 DECODING FUNCTIONS

5.7.1.1 INIT_GRIBHDR

Allocates storage for GRIB Header and its *entire_msg* and initializes every attribute. For some of the functions mentioned below, this function must be the first GRIB function called. This function needs to be called only one time in a program. The form is:

```
int      init_gribhdr (ppgrib_hdr, errmsg)
```

Where:

<<*Output* GRIB_HDR **ppgrib_hdr:

GRIB Header structure, Null upon entry. Returns pointing to a newly created storage. Its attribute '*entire_msg*' will point to a block of size indicated in '*abs_size*' (initially set to DEF_MSG_LEN bytes, see *grib.h*). '*entire_msg*' may later be expanded by other functions if required, but '*abs_size*' must be updated to the expanded byte length.

<<*Output* char *errmsg: Empty array, returned filled if error occurred

RETURNS:

0 — No error; storage for GRIB header and its *entire_msg* array created and cleared. *msg_length* and all section lengths are set to zero, all section pointers are Null. *abs_size* is set to DEF_MSG_LEN; 'shuffled' flag is set to zero.

1 — Failed, see *errmsg*

5.7.1.2 FREE_GRIBHDR

Frees up storage of GRIB Header structure and all its attributes. When *init_gribhdr* is called, this function must be the last function called in the program. The form is:

GRIB USER MANUAL

```
void free_gribhdr (ppgrib_hdr)
```

Where:

<<Output GRIB_HDR **ppgrib_hdr: GRIB Header structure whose storage is released.

RETURNS: None

5.7.1.3 INIT_DEC_STRUCT

Initializes the four internal Decoder structures. This function is mandatory when `grib_dec` is to be used. The form is:

```
void init_dec_struct ( pds, gds, bms, bds_head)
```

Where:

<<Output PDS_INPUT *pds: Internal PDS structure to be initialized

<<Output grid_desc_sec *gds: Internal GDS structure to be initialized

<<Output BMS_INPUT *bms: Internal BMS structure to be initialized

<<Output BDS_HEAD_INPUT *bds_head: Internal BDS struct to be initialized

RETURNS: None

5.7.1.4 INIT_ENC_STRUCT

Initializes structures `DATA_INPUT` and `GEOM_IN`. This function call is mandatory when functions `ld_enc_config`, `ld_enc_ieeeff`, `ld_enc_ffinfo` are used. The form is:

```
void init_enc_struct (data_input, geom_in, user_input)
```

Where:

<<Output DATA_INPUT *data_input: Encoder structure to be initialized

<<Output GEOM_IN *geom_in: Encoder structure to be initialized

<<Output USER_INPUT *user_input: Encoder structure to be initialized

RETURNS: None

5.7.2 DECODING

5.7.2.1 GRIB_DEC

This function decodes a Gridded Binary (GRIB Edition 1) format message. The function `int_gribhdr` must be called before `grib_dec` is used. The form is:

GRIB USER MANUAL

```
int grib_dec (curr_ptr, pds, gds, bds_head, bms, ppgrib_data,
errmsg)
```

Where:

<i>Input</i> >> char *curr_ptr:	Pointer to block containing GRIB message to decode
<< <i>Output</i> PDS_INPUT *pds:	To be filled with decoded Product Definition Section information
<< <i>Output</i> grid_desc_sec *gds:	To be filled with decoded Binary Data Section information
<< <i>Output</i> BDS_HEAD_INPUT *bds_head:	To be filled with decoded Binary Data Section information
<< <i>Output</i> BMS_INPUT *bms:	To be filled with decoded Bitmap Section information
<< <i>Output</i> float **ppgrib_data:	Points to NULL upon entry. Upon successful exit, points to newly allocated (malloc) Float array filled with unpacked and restored data
<< <i>Output</i> char *errmsg:	Empty array, returned filled if error occurred

RETURN CODE:

- 0 — Success, **ppgrib_data points to block with unpacked, restored data
- 1 — Fail: first 4 bytes of curr_ptr is not 'GRIB'
- 2 — Fail: last 4 bytes of curr_ptr is not '7777'
- 3 — Fail: not GRIB Edition 1
- 4 — Fail: unknown projection type

5.7.2.2 LD_DEC_LOOKUP

This function reads in the information from an external Lookup table (i.e., g1tab_2.1). This information is used to convert from GRIB Code Numbers to descriptive information about the field decoded. The form is:

```
int ld_dec_lookup (lookup_fn, errmsg)
```

Where:

GRIB USER MANUAL

Input>> char *lookup_fn: Name of Lookup file from which to read (i.e., /abspath/g1tab_128_2.1)

<<*Output* char *errmsg: Empty array, returned filled if error occurred

RETURN CODE:

- 0 — Successful, the four database tables filled
- 1 — File open error or error/eof while reading

5.7.2.3 GRIB_SEEK

This function searches the input file starting at the given offset for a GRIB message. If found, return it in GRIB_HDR structure. The form is:

```
int grib_seek (InFile, offset, Read_Index, gh, errmsg)
```

Where:

Input>> char *InFile Name of input file to search for message

<<*Input and Output*>> long *offset: Number of bytes to skip from the beginning of file. Gets updated upon leaving to absolute #bytes from beginning of file to beginning of message found.

Input>> int Read_Index: If set, only proceed if 'GRIB' starts exactly at the given byte offset

<<*Output* GRIB_HDR *gh: Empty upon entry to hold the Message found and its info

<<*Output* char *errmsg: Empty array, only filled if error occurred

RETURN CODE:

- 0 — No errors, may or may not have a valid message;

If no Msg was Found:

- a) errmsg will hold the Warning msg

If a valid Msg was Found:

- a) long *offset: if successful, gets updated to absolute beginning of Msg

GRIB USER MANUAL

b) Structure GRIB_HDR holds its info entire_msg: Is assigned to newly allocated (Malloc) unsigned char * array to hold entire message.

msg_length: size of entire_msg array in bytes.

ids_len, pds_len, gds_len, bms_len, bds_len, eds_len: Size of each defined sections in bytes.

ids_ptr: Points to message's Ident Data Sect

pds_ptr: Points to message's Prod Defn Sect

gds_ptr: Points to message's Grid Defn Sect

bms_ptr: Points to message's Bitmap Defn Sect

bds_ptr: Points to message's Binary Data Sect

eds_ptr: Points to message's End Data Sect

c) errmsg remains empty

1 — fseek/fread error, all ptrs in GRIB_HDR set to null; errmsg filled

2 — Got end of file, all ptrs in GRIB_HDR set to null; errmsg filled

3 — Null entire_msg pointer; errmsg filled

4 — Unable to open input file; errmsg filled

5.7.3 ENCODING

5.7.3.1 GRIB_ENC

`grib_enc` encodes a GRIB Edition 1 message using the three input internal structures (`DATA_INPUT`, `USER_INPUT`, `GEOM_IN`), and the Floating point data array. It is valid for Float array to be null if `GRIB_HDR` shows that it contains a predefined BDS. In that case, `grib_enc` exits with a non-error status.

```
int grib_enc (Data_Input, User_Input, Geom_In, pfData_Array,
             gh, errmsg)
```

Where:

<i>Input</i> >> DATA_INPUT Data_Input	Structure containing input field information.
<i>Input</i> >> USER_INPUT User_Input	Structure containing encoder configuration data.
<i>Input</i> >> GEOM_IN Geom_In	Structure containing grid geometry description.
<i>Input</i> >> float *pfData_Array	

GRIB USER MANUAL

Array of float data to be packed and stored in the Binary Data Section. Float array may be Null if the GRIB Header already contains a Binary Data Section in its attribute 'entire_msg'. That case is referred to as the 'Shuffle Mode' which results in the encoder creating only the sections that are not already in entire_msg.

<<Input and Output>> GRIB_HDR *gh

Pre-allocated (malloc) structure used to hold the encoded GRIB message and its information. It contains a large array to hold the encoded message, pointers to each of the Sections along with their length, and a flag 'shuffled' that determines how the message is encoded. If 'shuffled' is zero upon entry, all six sections will be created and array (float *pfData_Array) must contain the float data. If 'shuffled' is set upon entry, there is already one or more sections in Entire_msg. Each of these pre-included sections will have a Non-Null pointer and a non-Zero length. The encoder will then only create the missing sections and append them at the end of the existing sections in array 'entire_msg', hence these sections may not be in the proper order expected by GRIB.

<<Output char *errmsg: Empty array, returned filled if error occurred

RETURN VALUE:

0 — No errors

GRIB_HDR is returned with the encoded message in 'entire_msg', w/ total message length in msg_length, w/ pointers to each defined GRIB Header Sections in ids_ptr, pds_ptr, gds_ptr, bms_ptr, gds_ptr, eds_ptr, and each section length in ids_len, pds_len, gds_len, bms_len, bds_len, eds_len. Note that the sections may not be in order if the 'shuffled' bit is set.

1 — Failed, msg in errmsg

5.7.3.2 LD_ENC_CONFIG

Fill the structure holding user's input from config_fn that is passed in by the user. The form is:

```
int      ld_enc_config (config_fn, User_Input, errmsg)
```

Where:

<i>Input</i> >> char *config_fn:	Name of file to load from
<<Output USER_INPUT *User_Input:	Filled with data read from file
<<Output char *errmsg:	Returned filled if error occurred

RETURN CODE:

GRIB USER MANUAL

- 0 — Success, file is read and closed, `user_input` is filled
- 1 — Error opening file, `errmsg` filled
- 2 — Failed to get all expected arguments, `errmsg` filled
- 3 — Error in file, `errmsg` filled

5.7.3.3 LD_ENC_IEEEF

This function loads the user's pre-allocated (`malloc`) float array with data from binary flat file passed in by user (i.e., FF*). The form is:

```
int      ld_enc_ieeeff (ieee_fn, farr, elements, errmsg)
```

Where:

```
Input>> char *ieee_fn: Name of IEEE Flat file (with full path) to read
<<Output float *farr:   Pre-allocated (malloc) array to store data from read file
Input>> int elements   Number of float elements to read from file
<<Output char *errmsg   Returned filled if error occurred
```

RETURN CODE:

- 0 — Success, file is read and closed, float `arr` is filled
- 1 — Error opening file, `errmsg` filled
- 2 — Failed to get all expected elements, `errmsg` filled
- 3 — Incoming float array is null, `errmsg` filled

5.7.3.4 LD_ENC_FFINFO

This function fills the `DATA_INPUT` structure from file whose name is passed in. The form is:

```
int      ld_enc_ffinfo (ieee_info_fn, Data_Input, errmsg)
```

Where:

```
Input>> char *ieee_info_fn   Name of config information file to load
<<Output DATA_INPUT *Data_Input  To be filled with data read from config file
```

GRIB USER MANUAL

<<Output char *errmsg Filled if error occurred

RETURN CODE:

- 0 — Success, file is read and closed, structure is filled
- 1 — Error opening file
- 2 — Failed to get all expected arguments
- 3 — Ferror

5.7.3.5 LD_ENC_GEOMFILE

This function fills the GEOM_IN structure from file whose name is passed. The form is:

```
int        ld_enc_geomfile (geom_fn, Geom_In, errmsg)
```

Where:

Input>> char *geom_fn Name of geom information file to load

<<Output GEOM_IN *Geom_In To hold geom information read from file

<<Output char *errmsg Returned filled if error occurred

RETURN CODE:

- 0 — Success, file is read and closed, structure is filled
- 1 — Error opening file, errmsg filled
- 2 — Failed to get all expected arguments errmsg filled
- 3 — Ferror; errmsg filled

5.7.3.6 MAP_PARM

This function searches the previously defined array of parameter definitions for the specified parameter name. If a match is found, the function returns the GRIB parameter and sub-parameter ID's, as well as the defined scale and offset that is required to convert the input data array to GRIB units. If no match is found, an error message (errmsg) is returned. The form is:

```
int    map_parm (parm_name, data_input, parm_scl, parm_ref, errmsg)
```

GRIB USER MANUAL

Where:

<i>Input</i> >> char *parm_name	Name of Parameter to look for in the array of parameter structures
<< <i>Input and Output</i> >> DATA_INPUT *data_input	Three of its attributes filled usParm_id, usParm_sub_id, nDec_sc_fctr)
<< <i>Output</i> float *parm_scl	Scale factor to convert data to GRIB unit
<< <i>Output</i> float *parm_ref	Reference to convert to data to GRIB unit

RETURN CODE:

- 0 — Success, DATA_INPUT, parm_scl and parm_ref filled
- 1 — Parameter not found, errmsg filled

5.7.3.7 MAP_LVL

This function searches the previously defined array of level type definitions for the specified level type. If a match is found, the function returns the GRIB level ID for the level type, as well as the defined scale and offset that is required to convert the level values to GRIB units. The form is:

```
int map_lvl (lvl_type, data_input, lvl_scl_fctr, lvl_reference, errmsg)
```

Where:

<i>Input</i> >> char *lvl_type	Level to look for in the array of Level structures
<< <i>Input and Output</i> >> DATA_INPUT *data_input	Structure holding data pertaining to current message required by the encoder. Three of its attributes get filled (usLevel_id, nLvl_1, nLvl_2)
<< <i>Output</i> float *lvl_scl_fctr	Used along with lvl_reference to convert the Level to GRIB unit.
<< <i>Output</i> float *lvl_reference	Used along with lvl_scl_fctr to convert the Level to GRIB unit.

GRIB USER MANUAL

<<Output char *errmsg Empty array, returned filled if error occurred

RETURN CODE:

0 — Success, DATA_INPUT filled, fbuff may have changed

1 — Parameter not found, errmsg filled

5.7.3.8 LD_ENC_LOOKUP

This function reads GRIB definitions from an external encoder table. These definitions are used to convert information about the parameter, level, model, and geometry of the field being encoded to GRIB Code numbers. The form is:

```
int ld_enc_lookup (lookup_fn, errmsg)
```

Where:

Input>> char *lookup_fn Name of Lookup file to read

<<Output char *errmsg Returned filled if error occurred

RETURN CODE:

0 — Successful, the following pre-defined arrays required for encoding GRIB messages are filled:

PARAM_DEFN db_parm_tbl[NPARAM * MAX_PARM_TBLS] (parameter info)

LVL_DEFN db_lvl_tbl[NLEV] (level info)

MODEL_DEFN db_md1_tbl[NMODEL] (model info)

GEOM_DEFN db_geom_tbl[NGEOM] (geometry info)

1 — File open error or error/eof while reading, errmsg filled.

5.7.4 USER CONVENIENCE FUNCTIONS

5.7.4.1 GRIBHDR2FILE

This function writes out the GRIB message stored in GRIB_HDR structure to an external file. If the 'shuffle' flag is set, write each individual section out, else write 'entire_msg' all at once. The form is:

```
int gribhdr2file (gh, fn, errmsg)
```

GRIB USER MANUAL

Where:

Input>> GRIB_HDR *gh Contains GRIB message to be printed.

Input>> char *fn Name of file to write to (includes absolute path)

<<*Output* char *errmsg Array returned empty unless error occurred

RETURN CODE:

no errors - GRIB file successfully created

error - errmsg is filled

5.7.4.2 APPLY_BITMAP

This function applies the bitmap to the float array. The input float array is expanded and filled with 'fill_value' in places where data points are missing. The form is:

```
int apply_bitmap (bms, pgrib_data, fill_value, bds_head, errmsg)
```

Input>> BMS_INPUT *bms: Pointer to the internal BitMap header structure. Bit set means datapoint is present, bit clear means datapoint is missing

<<*Input and Output*>> float **pgrib_data: Pointer to Data that was unpacked from BDS's bitstr. Incoming size is bms->ulbits_set or (ROW*COL - #missingpts) elements.

Input>> float fill_value: Float value used for missing datapoints in expanded array.

<<*Output* BDS_HEAD_INPUT *bds_head: Attribute 'ulGrid_size' to be updated.

<<*Output* char *errmsg: Empty array that's returned filled if error occurred

GRIB USER MANUAL

RETURN CODE:

- 0 — Success. Float `**pgrib_data` probably have been expanded, OR redefined bitmap used, no action taken (float array unchanged).
- 1 — NULL bitmap encountered, `errmsg` filled
- 2 — Error allocating (Malloc) space for data array, `errmsg` filled
- 3 — Tried to access more than available in message, `errmsg` filled
- 4 — No bits set in BMS, `errmsg` filled

5.7.4.3 DISPLAY_GRIBHDR

This function does a byte dump for each of the defined GRIB Sections in the GRIB message currently stored in the GRIB Header structure. The function `init_gribhdr` must be called to use this function. The form is:

```
void display_gribhdr (gribhdr)
```

Where:

```
Input>> GRIB_HDR *gribhdr:   GRIB header info to be printed to standard
output
```

RETURN CODE: None

5.7.4.4 HDR_PRINT

This function prints a specified number of bytes from the block provided. It does not require that the Debug flag be set. The form is:

```
void  hdr_print (title, block, bytestoprint)
```

Where:

```
Input>>  char *title:           Title string to print

Input>>  unsigned char *block:   Block with content to print

Input>>  int bytestoprint:       Number of bytes to print
```

RETURN CODE: None

GRIB USER MANUAL

5.7.4.5 MAKE_DEFAULT_GRBFN

This function builds and returns a default filename for current message to be encoded using the information from structures DATA_INPUT and USER_INPUT. The form is:

```
void make_default_grbfn (DATA_INPUT di, USER_INPUT ui, char
*default_fn)
```

Where:

Input>> DATA_INPUT di: Contains information of msg to be encoded

Input>> USER_INPUT ui: Contains the required chCase_id

<<*Output* char *default_fn: Empty string at least 42 characters in length

RETURN CODE: None

<p>NOTE: default_fn string contains name with format Mid_Gid_yyyymmddhhtau_PIdx_Lid.lv11.c.grb</p>

5.7.4.6 MAKE_GRIB_LOG

Produces debug file GRIB.log from the GRIB message in the GRIB Header. To get the greatest amount of detail from the log, the function ld_dec_lookup must have been called previously to this function. The form is:

```
int make_grib_log (input_fn, lookup_fn, msg_length, offset,
pds, gds, bds, bms, grib_data, errmsg)
```

Where:

Input>> char *input_fn: Name of input GRIB file

Input>> char *lookup_fn: Name of Lookup file, null if not used

Input>> unsigned long msg_length: Total length of GRIB message

Input>> long offset: Starting location of GRIB message in bytes

Input>> PDS_INPUT pds: Product definition section structure

Input>> grid_desc_sec gds: Grid description section structure

Input>> BDS_HEAD_INPUT bds: Binary data section header structure

GRIB USER MANUAL

Input>> BMS_INPUT bms: Bit map definition section structure

Input>> float *grib_data: Array of decoded data

<<*Output* char *errmsg Text error message. NULL if function successful.

ACCESSES GLOBAL VARS:

int UseTables; Set to one if decoder table used

CTRS_DEFN db_ctr_tbl[NCTRS]: Predefined array holding Originating Center information

PARAM_DEFN db_parm_tbl [MAX_PARM_TBLS * NPARAM]:

Predefined array of Parameter information

LVL_DEFN db_lvl_tbl [NLVL]: Predefined array of Level information structure

MODEL_DEFN db mdl_tbl [NMODEL]: Predefined array of Model information structure

GEOM_DEFN db_geom_tbl [NGEOM]: Predefined array of Geometry information structure

RETURN CODE:

0 — No errors, file GRIB.log has been created

1 — Error, errmsg filled

5.7.4.7 PRT_INP_STRUCT

This function prints the content of the Internal GRIB structures. The form is:

```
void prt_inp_struct (pds, gds, bms_input, bds_head_input,
ppfarr)
```

Where:

Input>> PDS_INPUT *pds: Internal Product Defn Section structure to print

Input>> grid_desc_sec *gds: Internal Grid Defn Section to print

Input>> BMS_INPUT *bms_input: Internal Bitmap Section to print

GRIB USER MANUAL

Input>> structure BDS_HEAD_INPUT *bds_head_input:

Internal 11-byte hdr of Binary Data Section to print

Input>> float **ppfarr: Unpacked & restored float data array to print

RETURN CODE: None

5.7.4.8 INIT_STRUCT

This function initializes structures DATA_INPUT and GEOM_IN. The form is:

```
void init_struct (generic, size)
```

Where:

<<*Output* void *generic: Address of block to be cleared out

Input>> size_t size: Size of block

RETURN CODE: None

5.7.5 STRUCTURES

The include files `input.h` and `grib.h` contain definitions for several structures used by the MEL GRIB Software Library functions. However, this document will only cover the structures that are intended for access by the user.

5.7.5.1 THE USER_INPUT STRUCTURE (FROM INPUT.H)

The USER_INPUT structure holds configuration information for the encoder run, including some variables that are kept constant for a given data set. The structure can be filled manually or from a file using the function `ld_enc_config()`. Each of the elements of the structure is defined in the file `input.h`, but some additional discussion is warranted.

Unsigned char chCase_id

This is used to specify an alphanumeric character that will be used in the name of each GRIB file encoded (see default file name), allowing the user to uniquely identify files produced by a certain run.

NOTE: If the default file name is not used, this parameter is not required.
--

GRIB USER MANUAL

Unsigned short usParm_tbl

This specifies which standard GRIB table is being used. The WMO has recently released version 3 of the standard table, but most centers are still using version 2. The file `$GRIB_ENV/tables/g1tab_2.0` defines the WMO standard GRIB table, version 2. There are no model or geometry definitions made in the standard table.

Unsigned short usSub_tbl

This specifies the version of the local table being used. The local table allows the user to define extensions to the standard WMO table, and therefore the local table version is relative to the standard GRIB table version. It is very important that strict version control be maintained on all tables as this information is used by the decoder to determine if it has the correct table to decipher what is in a message. See the discussion on GRIB Table Management in **Section 5.6**.

Unsigned short usCenter_id

This indicates the GRIB code for the originating center as defined in Table 0 of the WMO GRIB standard. Definitions of 128 and above are left undefined by the WMO and can be used for local definitions.

Unsigned short usCenter_sub

This parameter is used to add another layer of classification to the identification of the originating center responsible for the encoding the message. The WMO defines official centers in Code Table 0 of the standard. The range 128-254 is left open for “National Use” but it is unclear how this is intended to be used and there is currently no official national list of originating centers. Therefore, the center sub-ID allows anyone to distinguish themselves as a sub-element of an identified center (e.g., NRL Monterey could be considered a sub-center of center # 058, Fleet Numerical Meteorology and Oceanography Center).

Unsigned short usGds_bms_id

This parameter represents the decimal equivalent of the byte defined by WMO Code Table 1, which is defined by the two leftmost bits as follows:

- bit 1: 0 → GDS omitted, 1 → GDS included
- bit 2: 0 → BMS omitted, 1 → BMS included
- bit 3-8: *Reserved by WMO and defined as 0*

The Grid Definition Section should typically be included. The Bit Map Section may or may not be included. In the configuration file, these bits are set individually and the function `ld_enc_config()` takes care of the conversion. If the `USER_INPUT` structure is filled manually, the decimal value of the `usGds_bms_id` will have to be computed based on the desired bit settings (i.e., a message with a GDS, but not BMS, gets a `usGds_bms_id = 128`).

GRIB USER MANUAL

Unsigned short usTrack_num

This is a MEL extension designed to allow tracking of data sets by assigning a unique integer to every dataset encoded and delivered from a resource site. This has not yet been implemented in the MEL Resource Site Software and this parameter should be set to 0.

Unsigned short usBDS_flag

This parameter represents the decimal equivalent of the byte defined by WMO Code Table 11, which is defined by the three leftmost bits as follows:

- bit 1: 0 → Grid point data, 1 → Spherical Harmonic Coefficients
- bit 2: 0 → Simple Packing, 1 → Complex or second-order packing
- bit 3: 0 → Floating point data, 1 → Integer data
- bit 4-8: *Used for complex packing only and defined as 0*

This flag is defaulted to 0 in the configuration file and should be left that way as the library currently does not support Spherical Harmonics or Complex Packing. Floating Point data and Integers are treated the same way in the library, so bit 3 is essentially irrelevant.

Unsigned short usBit_pack_num

This allows the user to specify how many bits are to be used per data point in packing the data. If it is set to 0, the encoder will determine the minimum number of bits required to pack the data, maintaining the resolution specified by the decimal scale factor. Unless the user thoroughly understands the packing method defined in the GRIB standard, it is *highly* recommended that this flag always be set to 0.

5.7.5.2 THE DATA_INPUT STRUCTURE (INPUT.H)

The DATA_INPUT structure holds all of the descriptive information about the field being encoded, (except for its geometry, which is specified in GEOM_IN).

Unsigned short usProc_id

This defines the ID of the model that created the data encoded in this message. The model IDs are defined in the local table. An undefined model is specified by setting this parameter to 255.

GRIB USER MANUAL

Unsigned short usGrid_id

This defines the ID of the geometry for data encoded in this message. The geometry IDs are defined in the local table. An undefined geometry is specified by setting this parameter to 255, which then requires that a geometry be defined in the GDS. Note that the converse is not true - a GDS can be included and usGrid_id can be set to a value other than 255; in fact, this is the recommended practice. Note also that setting this parameter does *not* imply that a GDS is present - that is handled by the usGds_bms_id in the USER_INPUT structure.

Unsigned short usParm_id

Unsigned short usParm_sub_id

The usParm_id holds the parameter identification code as defined in WMO Code Table 2. Any value greater than 128 implies a locally defined parameter. The MEL GRIB Software Library has defined an extension to WMO **GRIB Edition 1** that provides for the definition of five sub-tables to be used that allows for an additional 1275 locally defined parameters. These Sub-tables are defined as 2-A through 2-E, and each one contains 255 definitions. A sub-table parameter is referenced by usParm_id being set to 250 through 254, indicating a pointer to a specific sub-table (A through E), and usParm_sub_id being set to the code definition within the sub-table. See the discussion on GRIB Table Management in **Section 5.6.2** for further explanation.

CAUTION: Take note of the units defined in the table; *data must be in these units prior to calling grib_enc.*

Unsigned short usLevel_id

The usLevel_id holds the level type identification code as defined in WMO Code Table 3 (refer to **Section 5.6**). Locally defined level types are allowed, but there is no reserved range in Table 3 for local definitions and there is no sub-table extension.

Int nLvl_1, Int nLvl_2

Level 1 gives the value for the level type or defines the top of a layer. Level 2 is used to define the bottom level if the level type is a layer. These values are always integer and must be in the units specified in Table 3 of the GRIB standard. If the level type is a level, nLvl1 will be encoded into two bytes and so its maximum value is 65535. If the level type is a layer, nLvl1 and nLvl2 will each be encoded into 1 byte, and therefore each have a maximum value of 255.

Note that a typical 19.5 m height surface can not be properly encoded using the WMO level type 105 = "Specified Height Level above ground" because the units of integer meters will truncate the 19.5 to 19. To encode a level of 19.5 m one must resort to level type 125, which is the high-precision version of type 105, and encode the 19.5 m

GRIB USER MANUAL

value as 1950 centimeters. Since the maximum allowable value is 65535 the high-precision level type 125 can only be used up to a height of 655.35 m.

Int nYear, nMonth, nDay, nHour, nMinute, nSecond

All of these refer to the reference or base time of the data set, *not* the forecast time or valid time.

Unsigned short usFcst_id, usFcst_per1, usFcst_per2

The parameter `usFcst_id` specifies the time unit for the forecast parameters and is defined in GRIB Table 4. The `usFcst_per1` specifies the time of the forecast in the units specified by `usFcst_id`. The `usFcst_per2` gives the time interval between analyses or forecasts when averaging.

Unsigned short usTime_range_id, ustime_range_avg, usTime_range_mis

The indicator refers to the codes defined in Table 5. A simple forecast field is a 0. Other values are explained in Table 5, as well as the meaning of the two parameters which follow it.

Int nDec_sc_fctr

The input data is scaled up by 10 to the Decimal Scale Factor (DSF) prior to encoding. The effect is to set the resolution of the data to the number of decimal places in DSF. (i.e., DSF = 2 means a resolution of .01) Note that this is NOT intended to be used for changing the units of the data.

5.7.5.3 THE GEOM_IN STRUCTURE (INPUT.H)

The `GEOM_IN` structure holds the definition of the geometry. Only the elements of the structure required by the encoder are discussed here. The latitude/longitude convention for all parameters is defined as follows:

- Latitude is in the range ± 90 degrees (+ is north)
- Longitude can be in the range ± 180 degrees (+ is east) or 0 to 360 degrees (increasing to the east)

Char prjn_name

String containing the projection name. Valid options are "lambert", "spherical", or "polar_stereo".

Long nx, ny

GRIB USER MANUAL

The number of columns (nx) and number of rows (ny) on the grid. Columns are generally referred to as north-south lines (meridians), rows as east-west lines (parallels).

Double x_int_dis, y_int_dis

The grid resolution in kilometers in the x (east-west) and y (north-south) directions. Need not be filled for spherical projection grids.

Double parm_1, parm_2, parm_3

The definition of the parm 1,2,3 elements depend on the projection of the grid (prj name), as defined below:

- "spherical": Parm 1 stores the grid resolution in degrees of Latitude.
 Parm 2 stores the grid resolution in degrees Longitude.
 Parm 3 is undefined and should be set to -1.
- "lambert": Parm 1 stores the northern-most latitude at which the projection cuts the earth surface.
 Parm 2 stores the southern-most latitude at which the projection cuts the earth surface.
 Parm 3 stores the longitude that is parallel to the y-axis of the grid.
- "polar_stereo": Parm 1 and 3 are undefined and should be set to -1. Parm 2 stores the longitude that is parallel to the y-axis of the grid.

Double first_lat, first_lon

GRIB requires that the latitude and longitude of the first data point, as defined by the 'scan' parameter below, be given as the reference point of the grid system. Latitude is in the range +/- 90 degrees (+ is North). Longitude can be in the range +/- 180 degrees (+ is east) or 0 to 360 degrees (increasing to the east).

Double last_lat, last_lon

GRIB requires that the latitude and longitude of the last data point, as defined by the 'scan' parameter below, also be included for the spherical projection. These parameters follow the same conventions as first_lat/first_lon, and are not used for the other grid types.

GRIB USER MANUAL

Unsigned Short scan

A short integer that indicates how the data is ordered on the grid system. Also, it implicitly specifies where the first data point is located on the grid, thereby defining the `first_lat/first_lon` parameters above. This parameter is stored as the decimal equivalent of an 8-bit binary number defined by the three highest order bits, as defined by **WMO Code Table 8**:

bit 1 (128): 0 → data scans in +i direction (to the right)

1 → data scans in -i direction (to the left)

bit 2 (64): 0 → data scans in -j direction (downward)

1 → data scans in +j direction (upward)

bit 3 (32): 0 → data scans across rows first, then columns

1 → data scans across columns first, then rows

As an example, consider a data set for which the first data point is in the upper right corner of the grid, and the data is stored by scanning down columns from top to bottom, moving from the right-most to the left-most column. In this case, bits 1 and 3 would be set resulting in a decimal scan value of 160.

usRes_flag

A short integer that indicates a number of conventions about the grid system being defined, based on **WMO Code Table 7**. This parameter stores the decimal equivalent of an 8-bit binary defined by the following bit positions:

bit 1 (128): 0 → direction increments not given

1 → direction increments given

bit 2 (64): 0 → earth assumed spherical with radius 6367.47 km

1 → earth assumed oblate spheroidal (see WMO manual)

bit 5 (8): 0 → vector components are defined as north and east

1 → vector components are defined as +i and +j

If the `GEOM_IN` structure is filled manually, the `usRes_flag` parameter must be entered as a decimal value (i.e., "72" if bits 2 and 5 are set). However, if the external geometry file is used, and `GEOM_IN` is filled using the `ld_enc_geom` function as in the examples, bits 2 and 5 are entered explicitly and the function will handle the

GRIB USER MANUAL

translation to a decimal value. The function `ld_enc_geom` will also set bit 1 automatically based on whether or not `x_int_dis/y_int_dis` are defined..

5.7.5.4 THE GRIB_HDR STRUCTURE (GRIB.H)

The `GRIB_HDR` structure holds the encoded message and all of its information. The encoder will put the encoded message in the attribute `entire_msg`, which is a large array of type `char`.

Char shuffled

When this flag is zero, it means that all of the defined GRIB sections are in the order defined by the GRIB Edition 1 format (Indicator Section, Product Definition Section, optional Grid Definition Section, optional Bit Map Section, then Binary Data Section followed by End Section). When this flag is set to 1, it means the defined sections may not be in the proper order. This parameter can be used to efficiently input and output GRIB messages without fully decoding them. However, it should only be used by advanced users that fully understand the structure of GRIB messages.

Long msg_length

This holds the total byte-length of the message currently in the GRIB header.

Long ids_len,pds_len,gds_len,bms_len,bds_len,eds_len

Total length in bytes of the each of the GRIB sections. The sum of all six section lengths must equal the `msg_length` parameter.

Long abs_size

Current size of buffer `entire_msg` in the GRIB header. It is initially allocated `DEF_MESG_LEN` number of bytes, but is expanded if necessary. `DEF_MESG_LEN` is normally much larger than the actual Message length.

Unsigned char *entire_msg

Points to an array of type `Unsigned char` and is used to hold the entire GRIB message. This buffer is created via the call to function `init_gribhdr` and its length is stored in attribute `abs_size`.

Unsigned char

***ids_ptr,*pds_ptr,*gds_ptr,*bms_ptr,*bds_ptr,*eds_ptr**

These are pointers to all of the Header Sections that are present within `entire_msg`. If the section is not included, the pointer will then point to null.

5.8 RELATED PROCESSING

Not Applicable

5.9 DATA BACKUP

Not Applicable

5.10 RECOVERY FROM ERRORS, MALFUNCTIONS, & EMERGENCIES

The MEL GRIB Software Library will return an error code to the calling program. It is the responsibility of the developer of the main program to test for these errors and take appropriate action. Many times, this will result in terminating the program.

5.11 MESSAGES

Error messages returned by the MEL GRIB Software Library are listed in **Appendix F**.

This page intentionally left blank

GRIB USER MANUAL

APPENDIX A. GRIB MESSAGE STANDARD

GRIB Edition 1 Message

Indicator Data Section (IDS)	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"> 71 82 73 66 </td> <td style="text-align: center;"> 00 12 122 </td> <td style="text-align: center;"> 01 </td> </tr> <tr> <td style="text-align: center;">1 2 3 4</td> <td style="text-align: center;">5 6 7</td> <td style="text-align: center;">8</td> </tr> </table>	71 82 73 66	00 12 122	01	1 2 3 4	5 6 7	8	GRIB	3194	1																								
	71 82 73 66	00 12 122	01																															
1 2 3 4	5 6 7	8																																
	Ascii string	Message length in bytes	Edition																															
Product Definition Section (PDS)	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"> 0 0 28 </td> <td style="text-align: center;"> 2 </td> <td style="text-align: center;"> 58 </td> <td style="text-align: center;"> 75 </td> <td style="text-align: center;"> 237 </td> <td style="text-align: center;"> 128 </td> <td style="text-align: center;"> 2 </td> <td style="text-align: center;"> 102 </td> </tr> <tr> <td style="text-align: center;">1 2 3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> <td style="text-align: center;">6</td> <td style="text-align: center;">7</td> <td style="text-align: center;">8</td> <td style="text-align: center;">9</td> <td style="text-align: center;">10</td> </tr> </table>	0 0 28	2	58	75	237	128	2	102	1 2 3	4	5	6	7	8	9	10	28	2	58	75	237	128	2	102									
	0 0 28	2	58	75	237	128	2	102																										
1 2 3	4	5	6	7	8	9	10																											
	PDS length in bytes	GRIB Tbl Version International Exchange	Originating Center Id FNMOC	Model Id NORAPS- PTMUGU	Grid Id Pt Mugu	GDS,BDS Flag GDS included but not BMS	Parameter & Unit id Pressure Reduced to MSL	Level Type MeanSea Lvl																										
Grid Definition Section (GDS)	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"> 0 0 </td> <td style="text-align: center;"> 97 </td> <td style="text-align: center;"> 7 </td> <td style="text-align: center;"> 1 </td> <td style="text-align: center;"> 0 </td> <td style="text-align: center;"> 0 </td> <td style="text-align: center;"> 1 </td> <td style="text-align: center;"> 0 0 </td> </tr> <tr> <td style="text-align: center;">11 12</td> <td style="text-align: center;">13</td> <td style="text-align: center;">14</td> <td style="text-align: center;">15</td> <td style="text-align: center;">16</td> <td style="text-align: center;">17</td> <td style="text-align: center;">18</td> <td style="text-align: center;">19</td> <td style="text-align: center;">20</td> <td style="text-align: center;">21</td> <td style="text-align: center;">22 23</td> </tr> </table>	0 0	97	7	1	0	0	1	0	0	0	0 0	11 12	13	14	15	16	17	18	19	20	21	22 23	0	97	7	1	0	0	1	0	0	0	0
	0 0	97	7	1	0	0	1	0	0	0	0 0																							
11 12	13	14	15	16	17	18	19	20	21	22 23																								
	Height	Year of Century	Month	Day	Hour	Minutes	Forecast Time Unit in Hours	Period of Time Analysis	Time Interval	Time Range Indicator	Number Included in Avg																							
Bit Map Section (BMS)	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"> 0 </td> <td style="text-align: center;"> 20 </td> <td style="text-align: center;"> 0 </td> <td style="text-align: center;"> 128 1 </td> <td style="text-align: center;"> ... </td> <td style="text-align: center;"> 41 </td> <td style="text-align: center;"> 42 </td> <td style="text-align: center;"> 43 44 </td> <td style="text-align: center;"> 45 </td> <td style="text-align: center;"> 46 </td> </tr> <tr> <td style="text-align: center;">24</td> <td style="text-align: center;">25</td> <td style="text-align: center;">26</td> <td style="text-align: center;">27 28</td> <td style="text-align: center;">29 to 40</td> <td style="text-align: center;">41</td> <td style="text-align: center;">42</td> <td style="text-align: center;">43 44</td> <td style="text-align: center;">45</td> <td style="text-align: center;">46</td> </tr> </table>	0	20	0	128 1	...	41	42	43 44	45	46	24	25	26	27 28	29 to 40	41	42	43 44	45	46	0	20	0	-1	--	-	-	--	-	-			
	0	20	0	128 1	...	41	42	43 44	45	46																								
24	25	26	27 28	29 to 40	41	42	43 44	45	46																									
	Number Missing fr. Avg	Century	SubTbl entry for Originating ctr	Decimal Scale Factor	Reserved	Flag indicating use of NRL extensions	Seconds	Tracking Id fordataset	Sub-tbl entry for Parameters & Units	Sub-tbl Version number																								
Binary Definition Section (BDS)	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"> 0 0 32 </td> <td style="text-align: center;"> 0 </td> <td style="text-align: center;"> 255 </td> <td style="text-align: center;"> 0 </td> <td style="text-align: center;"> 0 61 </td> <td style="text-align: center;"> 0 51 </td> <td style="text-align: center;"> 0 113 72 </td> <td style="text-align: center;"> 129 238 36 </td> </tr> <tr> <td style="text-align: center;">1 2 3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> <td style="text-align: center;">6</td> <td style="text-align: center;">7 8</td> <td style="text-align: center;">9 10</td> <td style="text-align: center;">11 12 13</td> <td style="text-align: center;">14 15 16</td> </tr> </table>	0 0 32	0	255	0	0 61	0 51	0 113 72	129 238 36	1 2 3	4	5	6	7 8	9 10	11 12 13	14 15 16	32	0	255	0	61	51	29000	-126500									
	0 0 32	0	255	0	0 61	0 51	0 113 72	129 238 36																										
1 2 3	4	5	6	7 8	9 10	11 12 13	14 15 16																											
	GDS length in bytes	No. of Vert. coord.parmam	PV/PL means neither is present	Data Repre- sentation Spherical	No. Pts x-axis	No. Pts y-axis	Latitude of first point (millidegrees)	Longitude of first point (millidegrees)																										
End Data Section (EDS)	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"> 128 </td> <td style="text-align: center;"> 0 152 88 </td> <td style="text-align: center;"> 129 191 68 </td> <td style="text-align: center;"> 0 200 </td> <td style="text-align: center;"> 0 200 </td> <td style="text-align: center;"> 64 </td> <td style="text-align: center;"> 0 0 </td> </tr> <tr> <td style="text-align: center;">17</td> <td style="text-align: center;">18 19 20</td> <td style="text-align: center;">21 22 23</td> <td style="text-align: center;">24 25</td> <td style="text-align: center;">26 27</td> <td style="text-align: center;">28</td> <td style="text-align: center;">29 32</td> </tr> </table>	128	0 152 88	129 191 68	0 200	0 200	64	0 0	17	18 19 20	21 22 23	24 25	26 27	28	29 32	128	39000	-114500	200	200	64	0												
	128	0 152 88	129 191 68	0 200	0 200	64	0 0																											
17	18 19 20	21 22 23	24 25	26 27	28	29 32																												
	Resolution & Component flags Dir increments given; Earth Spherical; U&V relative to grid in +I & +J	Latitude of last point (millidegrees)	Longitude of last point (millidegrees)	I-direction Increment (millidegrees)	J-direction Increment (millidegrees)	Scanning mode Pts scan in +I, +J dir; Adjacent pts in I-dir are consecutive	Reserved																											
End Data Section (EDS)	Not included in this message																																	
	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"> 0 12 50 </td> <td style="text-align: center;"> 0 </td> <td style="text-align: center;"> 0 0 </td> <td style="text-align: center;"> 68 39 85 198 </td> <td style="text-align: center;"> 8 </td> <td style="text-align: center;"> 0 0 </td> </tr> <tr> <td style="text-align: center;">1 2 3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5 6</td> <td style="text-align: center;">7 8 9 10</td> <td style="text-align: center;">11</td> <td style="text-align: center;">12 3122</td> </tr> </table>	0 12 50	0	0 0	68 39 85 198	8	0 0	1 2 3	4	5 6	7 8 9 10	11	12 3122	3122	0	0	10069.773438	8	0	0														
0 12 50	0	0 0	68 39 85 198	8	0 0																													
1 2 3	4	5 6	7 8 9 10	11	12 3122																													
	BDS length in bytes	Flag Grid pt data; Simple packing; Float values	Binary Scale Factor	Reference Value also minimum value of data	Num of bits per packed value	*** Actual packed data Zero filled to an even number of octets																												
End Data Section (EDS)	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;"> 55 55 55 55 </td> </tr> <tr> <td style="text-align: center;">1 2 3 4</td> </tr> </table>	55 55 55 55	1 2 3 4	7777																														
	55 55 55 55																																	
1 2 3 4																																		
	Ascii string																																	

This page intentionally left blank

GRIB USER MANUAL

APPENDIX B. MEL GRIB SOFTWARE LIBRARY INVENTORY

NOTE: This listing is not meant for configuration control. Some distributed files may have different dates.

Main directory

Name	Date	Time	Size (KB)
Clean	Feb 23	10:04	2159
DISCLAIMER	Sep 9	1997	3009
Install	Dec 16	12:50	3539
README.DOC	Dec 16	13:31	3927
VERSION	Dec 16	13:28	147
Bin	Jun 8	9:19	9
Config	Jun 8	9:19	32
config.os	Apr 23	9:33	5491
Data	Jun 8	9:19	4096
Doc	Jun 8	9:19	75
Include	Jun 8	9:19	147
Lib	Jun 8	9:19	9
Libsrc	Jun 8	9:31	4096
Run	Jun 8	9:19	9
Src	Jun 8	9:19	84
Tables	Jun 8	9:19	4096

Bin

Name	Date	Time	Size (KB)
------	------	------	-----------

Config

Name	Date	Time	Size (KB)
encoder.config	Feb 12	11:38	759

data

Name	Date	Time	Size (KB)
058.223.dwpt_dprs.isbr_lvl.1000.0.1997081200.012	Aug 27	1997	11910
058.223.geop_ht.isbr_lvl.100.0.1997081200.000	Aug 27	1997	19794
058.223.geop_ht.isbr_lvl.100.0.1997081200.012	Aug 27	1997	19794
GRIB.test1	Apr 23	9:25	15309
GRIB.test2	Apr 23	9:31	16863
GRIB0797.tar	Jul 21	1997	30720
IEEE.input	Jul 21	1997	12444
encoder_ex2.geom	Feb 12	10:09	810
encoder_ex2.info	Feb 12	10:04	691
encoder_ex3.list	Aug 26	1997	148
getgribieee.list	Aug 27	1997	165
ieee.pres.msl.2.0.1997070100.000	Aug 13	1997	12444
ieee.pres.msl.2.0.1997070100.012	Aug 13	1997	12444
ieee.wnd_ucmp.ht_sfc.10.0.1997070100.000	Aug 13	1997	12444
ieee.wnd_ucmp.ht_sfc.10.0.1997070100.012	Aug 13	1997	12444
ptmugu_61x51.geom	Feb 12	10:10	826

GRIB USER MANUAL

doc

Name	Date	Time	Size (KB)
Interface.doc	Apr 23	14:23	27021
Method.all	Apr 23	14:21	125558
run_examples.doc	Aug 29	1997	3214

include

Name	Date	Time	Size (KB)
dprints.h	Feb 20	9:43	3299
grads.h	Apr 16	10:41	5469
grib.h	Apr 23	9:14	21238
grib_lookup.h	Dec 10	13:50	6266
gribfuncs.h	Feb 22	16:44	4117
gsv5d.h	Nov 20	1997	2993
input.h	Jun 19	1997	3717
isdb.h	Dec 22	9:24	5447

lib

Name	Date	Time	Size (KB)
------	------	------	-----------

libsrc

Name	Date	Time	Size (KB)
FTP_getfile.c	Apr 15	13:27	4958
Makefile	Feb 23	8:48	3144
apply_bitmap.c	Apr 20	11:00	6380
changes.log	Apr 23	9:58	6850
display_gribhdr.c	Jun 25	1997	5132
gbyte.c	Sep 16	1997	14124
grib_dec.c	Feb 22	15:46	7850
grib_enc.c	Feb 20	9:40	19620
grib_seek.c	Dec 15	8:29	18915
gribgetbds.c	Feb 18	18:51	10416
gribgetbms.c	Jun 25	1997	4966
gribgetgds.c	Feb 18	21:00	22326
gribgetpds.c	Apr 23	9:16	9110
gribhdr2file.c	Dec 15	9:17	5263
gribputbds.c	Nov 4	1997	7433
gribputgds.c	Feb 26	10:15	40929
gribputpds.c	Apr 23	9:18	16804
hdr_print.c	Jun 25	1997	1617
init_dec_struct.c	Apr 23	8:54	2183
init_enc_struct.c	Jun 25	1997	1612
init_gribhdr.c	Nov 4	1997	8458
init_struct.c	Aug 27	1997	1167
ld_dec_lookup.c	Feb 18	19:39	26458
ld_enc_input.c	Feb 10	9:00	21582
ld_enc_lookup.c	Aug 27	1997	17126
ld_grib_origctrs.c	Apr 15	11:12	4730
make_default_grbfn.c	Aug 27	1997	2353
make_grib_log.c	Apr 23	9:21	29960
map_lvl.c	Dec 15	8:53	3403
map_parm.c	Dec 15	9:03	4399
pack_spatial.c	Nov 13	1997	17537
prt_badmsg.c	Feb 24	10:07	9514
prt_inp_struct.c	Apr 23	9:11	14870
upd_child_errmsg.c	Jun 25	1997	1652

GRIB USER MANUAL

run

Name	Date	Time	Size (KB)
------	------	------	-----------

src

Name	Date	Time	Size (KB)
decoder_ex	Jun 8	9:31	47
encoder_ex	Jun 8	9:31	92
Getgribieee	Jun 8	9:31	68
Gribsimp	Jun 8	9:31	4096

decoder_ex

Name	Date	Time	Size (KB)
Makefile	Feb 23	8:58	920
decoder_ex.c	Apr 23	11:22	7119

encoder_ex

Name	Date	Time	Size (KB)
Makefile	Feb 23	9:21	1720
encoder_ex1.c	Jan 22	10:22	8346
encoder_ex2.c	Dec 23	14:05	5824
encoder_ex3.c	Feb 23	11:19	13202

getgribieee

Name	Date	Time	Size (KB)
Makefile	Feb 23	10:20	912
changes.log	Feb 23	8:41	81
getgribieee.c	Feb 26	8:21	7023

gribsimp

Name	Date	Time	Size (KB)
Makefile	Apr 16	11:17	2653
changes.log	Apr 23	9:59	1162
e_h_time.c	Nov 13	1997	2677
grad_boundary_box.c	Nov 13	1997	16398
gribsimp.c	Apr 16	9:38	62105
h_e_time.c	Jul 22	1997	2526
ld_grad_msg.c	Aug 27	1997	33003
ld_v5d_msg.c	Dec 17	8:36	29100
make_grad_files.c	Apr 9	9:50	36465
make_v5d_file.c	Apr 14	12:14	33290
reg_trfm.c	Jul 22	1997	31809

This page intentionally left blank

APPENDIX C. GRIB EXTENSIONS

The GRIB standard was selected for use by the MEL project because of its relatively widespread international acceptance, its portability across all platforms, and its efficient and flexible packing scheme for gridded data. However, there are a few shortcomings of the GRIB format that can be addressed through the addition of locally defined extensions to the PDS of each GRIB message. The GRIB standard allows the PDS to be of variable length for just this purpose. As long as the required octets of the PDS are used in the standard way, a GRIB message with extensions to the PDS is still considered fully compliant with the WMO GRIB standard.

The WMO GRIB standard defines 28 octets for the default, and required PDS section of a GRIB message. Octets 29 - 40 are currently undefined, but still reserved by the WMO for future use, and need only be present if local octet definitions are made beyond octet 40. The MEL GRIB extensions occupy octets 41 - 46 of the PDS section, and are described in detail below.

Octet 41: Extension Flag

This parameter, along with the absolute length of the Product Definition Section (PDS), is used to determine if the message being decoded contains recognized extensions. The Flag is checked against the value of the constant defined in `grib.h`, "EXTENSION_FLAG", which is currently set to 99. The value of EXTENSION_FLAG should not be modified without first consulting the developers of this software.

The current check for the GRIB extensions defined in this software library is:

```
(PDS.octet41 == EXTENSION_FLAG)    &&    (PDS.length == 46)
```

Octet 42: Seconds of the Reference Time

The PDS defines the year, month, day, hour, and minute of the reference time of the data set in octets 13 through 17. For some data sets, the reference time is specified to the precision of seconds, and so this extension allows this specification.

GRIB USER MANUAL

Octets 43-44: Tracking ID

This 2 byte parameter is intended to provide a unique integer to define the data set being delivered. The idea is to allow users of the message to reference a unique ID should they need to contact the originating center concerning any problems with the data set. This unique ID would allow the originating center to efficiently determine the exact data set delivered. This feature is currently not used by the MEL Resource Site Software.

Octet 45: Parameter Sub-ID

Perhaps the biggest limitation of the GRIB standard is that it only allows for 255 unique parameter definitions, of which 128 are reserved for use by the WMO. This is specified in octet 9 of the PDS, and is referred to as the Parameter ID. The Parameter Sub-ID allows for the definition of an additional 1275 local parameters. This is accomplished through the definition of five parameter sub-tables, each containing up to 255 definitions (code 000 is reserved and can not be used). The sub-tables are labeled A through E, and are referenced in the GRIB message by setting the parameter ID (octet 9) to 250 through 254, (i.e., 250 points to sub-table A, 251 points to sub-table B, etc.). The parameter contained in the message is then defined by setting the parameter sub-ID (octet 45) to the sub-table entry for that parameter. As an example, consider the parameter "white cap probability" which is defined as parameter 78 of sub-table A at the NRL, Monterey MEL Resource Site. In the extended PDS section, this parameter would be represented with octet 9 set to 250 (pointer to sub-table A), and octet 45 set to 78.

Note that if a decoder that does not have these extensions implemented attempts to decode a message with a parameter from a sub-table, it will only return the parameter ID and hence will not be able to determine the parameter name. For this reason, sub-table parameter definitions are used sparingly. However, any decoder can still decode a message with a sub-table parameter, and obtain the data, model, geometry, and level information. It is only the parameter name that is indecipherable.

Octet 46: Local Table Sub-ID

This parameter allows encoding centers to maintain version control on their locally defined encoding tables. The WMO specifies the version of their standard tables in octet 4 of the PDS, and states that values 128-254 are reserved for local use. However, the problem with this approach is that since the local and WMO table versions are combined, there is no way for a decoder to determine the WMO table version if a local table version is included. Therefore, it was decided to split the version information and provide a separate octet for local table version.

APPENDIX D. RUNNING GRIB EXAMPLES

D.1 RUNNING 'DECODER_EX'

It is recommended that this example be run in directory `$GRIB_ENV/run`. From the UNIX prompt, type:

```
decoder_ex1
```

Input file:

```
$GRIB_ENV/data/GRIB0797.tar — file holding the GRIB messages to decode.
```

Output file:

```
./decoder_ex.output — a large text file displaying every datapoint value of each decoded message.
```

D.2 RUNNING 'GETGRIBIEEE'

Run this example in directory `$GRIB_ENV/run`. From the UNIX prompt, type:

```
$GRIB_ENV/bin/getgribieee $GRIB_ENV/data/getgribieee.list
```

Input file:

```
$GRIB_ENV/data/getgribieee.list
```

`getgribieee.list` contains a list of input GRIB files to process

Here, the list file contains:

```
$GRIB_ENV/data/058.223.geop_ht.isbr_lvl.100.0.1997081200.000  
$GRIB_ENV/data/058.223.geop_ht.isbr_lvl.100.0.1997081200.012  
$GRIB_ENV/data/058.223.dwpt_dprs.isbr_lvl.1000.0.1997081200.012
```

Output files:

Three files hold IEEE data of input GRIB files

```
058.223.geop_ht.isbr_lvl.100.0.1997081200.000.IEEE  
058.223.geop_ht.isbr_lvl.100.0.1997081200.012.IEEE  
058.223.dwpt_dprs.isbr_lvl.1000.0.1997081200.012.IEEE
```

GRIB USER MANUAL

NOTE: These IEEE files *do not* contain the 4-byte Header and Trailer, just the floating-point data. They may also be compared with the binary files created by `gribsimp` option `-o` using the same input GRIB files.

D.3 RUNNING 'ENCODER_EX1'

Run this example in directory `$GRIB_ENV/run`. At the UNIX prompt, type:

```
encoder_ex1
```

Input file:

```
$GRIB_ENV/data/IEEE.input — the float data used for encoding
```

Output file:

```
075_237_1997070100012_0011_105.00002.0.grb — the encoded GRIB  
message
```

D.4 RUNNING 'ENCODER_EX2'

Run this example in directory `$GRIB_ENV/run`. From the UNIX prompt, type:

```
encoder_ex2
```

Input files:

```
$GRIB_ENV/config/encoder.config — holds the Encoder configuration information
```

```
$GRIB_ENV/data/encoder_ex2.geom — holds the Geometry information
```

```
$GRIB_ENV/data/encoder_ex2.info — holds information on message to encode
```

```
$GRIB_ENV/data/IEEE.input — holds the float data used for encoding
```

Output file:

```
075_237_1997070100012_0011_105.00002.1.grb — the encoded GRIB  
message
```

GRIB USER MANUAL

D.5 RUNNING 'ENCODER_EX3'

This program expects three command line arguments: Model type, Geometry name, and the List filename, respectively.

```
encoder_ex3 NORAPS2 ptmugu_61x51 $GRIB_ENV/data/encoder_ex3.list
```

Input files:

`$GRIB_ENV/config/encoder.config` — file holding the configuration information for the encoder.

`$GRIB_ENV/data/'$GEOM'.geom` — Geometry information file. Since the command line specifies geometry `ptmugu_61x51`, the program will automatically look for file `$GRIB_ENV/data/ptmugu_61x51.geom`.

`$GRIB_ENV/data/encoder_ex3.list` — file contains the names of the input IEEE files to be used for encoding. Each entry appears on a single line and may *not* contain a path. The program automatically adds the path `$GRIB_ENV/data/` in front of each file name.

Here is the listing of `$GRIB_ENV/data/encoder_ex3.list`:

```
ieee.pres.msl.2.0.1997070100.000
ieee.pres.msl.2.0.1997070100.012
ieee.wnd_ucmp.ht_sfc.10.0.1997070100.000
ieee.wnd_ucmp.ht_sfc.10.0.1997070100.012
```

IEEE input files

These files hold the IEEE data used to encode each message. The filename format is "ieee."parmname.lv1type.lv11.lv12.yyyyymmddhh.tau, as listed in the List file. All listed files must reside in directory `$GRIB_ENV/data/`.

Output files:

From the four input "ieee." entries in the Listing files, four GRIB messages are encoded and stored in separate files =

```
016_054_1997070100000_0001_102.00002.3.grb
016_054_1997070100000_0033_105.00010.3.grb
016_054_1997070100012_0001_102.00002.3.grb
016_054_1997070100012_0033_105.00010.3.grb
```

This page intentionally left blank

APPENDIX E. GRIB SOFTWARE CONVENTIONS

E.1 DEFAULT FILENAME FOR ENCODED GRIB MESSAGES

The MEL GRIB Software defines a default naming convention as follows:

```
MMM.GGG.yyyyymmddhhtau_PPPX.lv11.c.grb
```

Where:

MMM : 3-digit model ID
GGG : 3-digit geom ID
YYYY : 4-digit year of reference date/time
mm : 2-digit month of reference date/time
dd : 2-digit day of reference date/time
hh : 2-digit hour of reference date/time
tau : 3-digit forecast
PPPX : 4-digit Parameter Index computed from Parmid & ParmSubid
Lid : 3-digit Level ID
lv11 : 5-digit Level
.c : 1-digit Case ID
.grb : 4-char string, as is

E.2 THE IEEE FILES

Throughout this document, many references to IEEE files have been made. In this context, these are 32-bit IEEE unformatted binary files that contain just the floating point data of a GRIB message. They DO NOT contain the 4-byte Header and Tail as in the FORTRAN IEEE files.

For a GRIB message with Rows by Cols dimension, an IEEE file of size (Rows * Cols * size of Float) bytes will be generated. In the GRIB library, the programs 'getgribieee' and 'gribsimp' can create these IEEE files.

The IEEE files generated by the MEL GRIB Software Library can be easily read by any C or FORTRAN program. Some simple examples are shown below.

The IEEE files can be read into a C program by simply opening the files in "binary read" (rb) mode and using the fread function to access the data. For example:

GRIB USER MANUAL

```
void main() {
    FILE *f1;
    float *fbuff;
    int rows, cols;

    rows = 61; cols = 51; /* depends upon the Geometry */
    f1 = (FILE *)fopen("IEEE.input", "rb");
    fbuff = (float *)malloc(sizeof(float) * rows * cols);
    if (fread ((void*)fbuff, sizeof(float), rows*cols, f1)
        != rows*cols)
    { /* error handling */ }
    /* data has been loaded in to float data array fbuff[0] through
    fbuff[(rows*cols) -1]
    */
    fclose(f1);
    free (fbuff);
}
```

Using FORTRAN-77, open the IEEE files as 'Unformatted' and 'Direct' access. Then read in one record at a time where the record length (RECL) depends upon the system. Here, RECL is set to 1 for SGI IRIX64 system (SunOS requires RECL to be 4). Consult your FORTRAN compiler's reference manual for more information on direct access reads.

```
PROGRAM                f77sample
...
real*4                fbuff(61 * 51)
character*10          IEEEfn/'IEEE.input'/

C                    Open the Input file, MODE: Unformatted, DIRECT access:
C                    Note: RECL value is 1 for IRIX64, 4 for SunOS
open (unit=2, file=IEEEfn, status='old',
      +          form='unformatted', access='direct', RECL=1)

C                    Read in 61*51 (or 3111) "real*4" elements, one at a
time
do 150 i=1,3111
read (2, REC=i) fbuff(i)
150   enddo
close (2)
C                    Data has been loaded into float array now
...
```

E.3 DECODED IEEE FILE NAMES (FROM GRIBSIMP)

The `-o` flag instructs `gribsimp` to generate a 32-bit IEEE unformatted binary data file for each message decoded. These output files contain the data only, organized as specified in the Grid Definition Section of the message. There is no 4-byte header or trailer like those used in FORTRAN IEEE files. The file naming convention used for decoded files is:

GRIB USER MANUAL

FFyymmddhhtau.PID.GID.LID.level

Where:

FF : 2-characters indicating a Flat File (IEEE). The second character may be changed to one of A/B/C/D/E to indicate the use of a sub-table identifier for this parameter.
YY : 2-digit year of reference date/time
mm : 2-digit month of reference date/time
dd : 2-digit day of reference date/time
hh : 2-digit hour of reference date/time
tau : 3-digit forecast period relative to the reference date/time
PID : 3-digit GRIB parameter code
GID : 3-digit GRIB grid definition code
LID : 3-digit GRIB level type code
level: 5-digit scaled integer of level 1 value

Example: FF97070100012.002.237.105.00010

This file contains the data from a message that is the 12-hr forecast field from a model run started on July 1, 1997 at 00Z. The message is for parameter 002 on level type 105 and Height of 00010 for grid 237.

E.4 DEFAULT DECODER TABLE FILE NAME CONVENTIONS

The following is the default decoder table file name used in `gribsimp`:

gEtab_CEN.{SUB_CEN}_X.Y

Where:

E : GRIB edition number (currently 1)
CEN : Originating Center ID
SUB_CEN : Originating Center SUB-ID (optional)
X : WMO GRIB table version number (currently 2)
Y : Local GRIB table version number

Example: 'g1tab_58_2.1'

This file contains the Decoder Tables for Center 58 using GRIB Edition 1 with version 2 of the WMO Code Tables, and version 1 of the locally defined tables at Center 58.

This page intentionally left blank

GRIB USER MANUAL

APPENDIX F. MEL GRIB SOFTWARE LIBRARY ERROR MESSAGES

The following are the MEL GRIB Software Library error messages that may be returned to the user:

```
expand_gribhdr: either GRIB_HDR or Entire_msg is Null
expand_gribhdr: failed to create new array ((newsize) bytes)
FTP_getfile: failed to open '(file_name)' for reading
FTP_getfile: Fail to read 2 args from '(file_name)'
FTP_getfile: failed to build FTP script
FTP_getfile: system call to ftp failed
FTP_getfile: '(file_name)' not avail on '(host_name)' in '(path_name)'
apply_bitmap: No bits set in bitmap. No data retrieved!!
apply_bitmap: Error mallocing (# of bytes) bytes
apply_bitmap: accessing more than (total bits set) elements in Grib_data[]
grib_dec: no 'GRIB' at beg. of this msg
grib_dec: no '7777' at end of this msg
grib_dec: not Grib Edition 1
grib_dec: unknown usData_type=(usData_type)
grib_enc: expecting non-null GRIB_HDR struct
grib_enc: failed to make storage for Internal Structs
grib_enc: <Create-All mode> No DataArray avail to encode
grib_enc: error Expanding Array to include Sect5 ((message_length) byte)
grib_enc: GribHdr Length/Ptr to sections are not consistent
grib_enc: <Create Missing Sect mode> No DataArray avail to encode Bds
grib_seek: expecting non-NULL Grib Hdr;
grib_seek: Cannot open input file '(Infile)'
grib_seek '(InFile)': Got fseek error to pos= '(pos)'
grib_seek '(InFile)': skip last '(nread)' bytes, too few for a Msg
grib_seek '(InFile)': No Grib msg found at offset= '(offset)'; check Index File
grib_seek '(InFile)': FSEEK error to pos+bytenum= '(pos+bytenum)'
grib_seek '(InFile)': failed to Expand entire msg to '(lMessageSize)' bytes
grib_seek '(InFile)': failed to read EntireMsg (sz='(lMessageSize)')
grib_seek '(InFile)': corrupt PDS len= '(pds_len)', Totlen='(msg_length)', drop
msg;
grib_seek '(InFile)': corrupt GDS len= '(gds_len)', Totlen='(msg_length)', drop
msg;
grib_seek '(InFile)': corrupt BMS len= '(bms_len)', Totlen='(msg_length)', drop
msg;
grib_seek '(InFile)': corrupt BDS len= '(bds_len)', Totlen='(msg_length)', drop
msg;
grib_seek '(InFile)': no 7777 found for msg at '(offset)', check indexfile
grib_seek '(InFile)': No Grib Msg found at IndexFile's offset = '(offset)';
Check Index File
gribgetbds: unrecognized packing algorithm
```

GRIB USER MANUAL

```
gribgetbds: BMS present, #datapts calculated '(num_calc)' not same as BMS's set
bits '(ulbits_set)'
gribgetbds: GDS present, #datapts calculated '(num_calc)' not same as BMS's grid
size '(ulGrid_size)'
gribgetbds: failed to malloc Grib_Data
gribgetbms: corrupted BMS, gds_flag set but totbits '(totbits)' != ulgrid_sz
'(ulGrid_size)'
gribgetgds: unknown datatype='(usData_type)'
gribhdr2file: GRIB_HDR message buffer is null, OR msg_length=0
gribhdr2file: Shuffle mode: Zero length encountered, quit
gribhdr2file: Unable to open '(fn)'
gribhdr2file: failed to Fwrite IDS to file
gribhdr2file: failed to Fwrite PDS to file
gribhdr2file: failed to Fwrite GDS to file
gribhdr2file: failed to Fwrite BMS to file
gribhdr2file: failed to Fwrite BDS to file
gribhdr2file: failed to Fwrite EDS to file
gribhdr2file: failed to write GH's entire Msg to file
gribputbds: Grib Header or its Entire_msg is NULL
gribputbds: Float array is Null, cannot proceed;
gribputbds: No FloatData avail and GribHdr has no BDS yet (ptr='(bds_ptr)'
len='(bds_len)')
gribputbds: failed to REALLOC entire msg to '(newsize)' bytes
gribputgds: grib header is null
gribputgds: MALlocated true Grib struct failed
gribputgds: Projection '(prjn_name)' unknown
gribputgds: failed to REALLOC entire msg to '(new_msgs)' bytes
create_inpLambert: ppvGDS_Proj_Input is null
create_inpPolar: ppvGDS_Proj_Input is null
create_inpLatlon: ppvGDS_Proj_Input is null
inp2grib_Lambert: the VOID *ppvGDS_Proj_Input block is null
inp2grib_PolarSt: Polar or pProjInp is null
inp2grib_Latlon: lLatlon_inp || pLatlon is null
gribputpds: failed storage for PDS_GRIB
gribputpds: failed to REALLOC entire msg to '(msg_length+sizeof(PDS_GRIB))' bytes
init_gribhdr: failed to create storage for GRIB_HDR
init_gribhdr: failed to create storage for GRIB_HDR's Msg
ld_dec_lookup: failed to open '(lookup_fn)'
ld_dec_lookup: got EOF/ERROR before PARM TABLE #'(sub)' info (Line '(LineRead)'
in '(lookup_fn)')
ld_dec_lookup: got EOF/ERROR before loading LEVEL info (Line '(LineRead)' in
'(lookup_fn)')
ld_dec_lookup: got EOF/ERROR before loading MODEL info Line '(LineRead)' in
'(lookup_fn)'
ld_dec_lookup: got EOF/ERROR before loading GEOM info Line '(LineRead)' in
'(lookup_fn)'
ld_enc_config: Failed to open '(config_fn)'
ld_enc_config: failed to extract arg from line: '(line)'
ld_enc_config: got ferror(infile)
ld_enc_config: Failed to load '(config_fn)' ('(linenum-1)'/'(num_expected)')
```

GRIB USER MANUAL

```
ld_enc_lookup: failed to open '(lookup_fn)'  
ld_enc_lookup: got EOF/ERROR before loading DBs PARM info ('(lookup_fn)':line  
'(LineRead)')  
ld_enc_lookup: got EOF/ERROR before loading LEVEL info (line '(LineRead)' in  
'(lookup_fn)')  
ld_enc_lookup: got EOF/ERROR before loading MODEL info (line '(LineRead)' in  
'(lookup_fn)')  
ld_enc_lookup: got EOF/ERROR before loading GEOM info (line '(LineRead)' in  
'(lookup_fn)')  
ld_grib_origctrs: got EOF/ERROR skipping over Header lines in '(fn)'  
ld_grib_origctrs: Invalid Ctr_id '(strGribCode)', LINE='(Line)'  
make_grib_log: failed to open 'GRIB.log'  
map_lvl: no '(lvl_type)' in db_lvl_tbl;  
map_parm: no '(parm_name)' in db_parm_tbl  
pack_spatial: invalid pt_cnt = '(pt_cnt)'  
pack_spatial: invalid bit_cnt = '(bit_cnt)'  
pack_spatial: Grid contains all NULLS  
pack_spatial: Calculated bit count OUT OF RANGE!!  
pack_spatial: Malloc failed pBitstream  
prt_badmsg: no 'GRIB' at beg. of this msg. Cannot continue.  
prt_badmsg: Message length too short (lMessageLen), cannot continue  
prt_badmsg: unknown usData_type (gds.head.usData_type)upd_child_errmsg: no Error  
msg avail!
```

This page intentionally left blank

APPENDIX G. ACRONYMS/ABBREVIATIONS

2-D..... Two Dimensional
3-D..... Three Dimensional
5-D..... Five Dimensional

A

ASCII..... American Standard Code for Information Interchange

B

BDS Binary Data Section
BMS Bitmap Section
BUFR..... Binary Universal Form for Representation of meteorological data

C-D

DMSO Defense Modeling and Simulation Office
DoD..... Department of Defense
DSF..... Decimal Scale Factor
dtg..... Date Time Group

E-F

FTP..... File Transfer Protocol

G

GDS..... Grid Definition Section
GrADS..... Grid Analysis and Display System
GRIB Gridded Binary representation of meteorological data

H

HTTP..... Hypertext Transfer Protocol

GRIB USER MANUAL

I

ID Identification
IEEE Institute of Electrical and Electronics Engineers

J-K-L

lvl..... level

M

m..... meter (e.g., 2-meter Dry Bulb temperature)
M&S Modeling and Simulation
MEL Master Environmental Library
MSEA Modeling and Simulation Executive Agent
MSMP Modeling and Simulation Master Plan

N

NRL..... Naval Research Laboratory
nrlmry Naval Research Laboratory, Marine Meteorology Division, Monterey, CA

O-P

PDS Product Definition Section

Q-R-S

SGI Silicon Graphics, Incorporated

T-U

tar tape archive
URL..... Universal Resource Locator
USD(A&T) Undersecretary of Defense for Acquisition and Technology

V-W

WMO..... World Meteorological Organization

X-Z

GRIB USER MANUAL

INDEX

- atmosphere iii
- Binary Definition Section 31, 32, 37, 38, 68, 69, 71,
77, 79, 81, 107, 108, 111
- Bit Map Section 9, 31, 32, 37, 38, 45, 68, 69, 77, 78,
80, 82, 88, 107, 108, 111
- buffer 32, 34, 36, 48, 51, 88, 108
- C 1, 5, 12, 13, 15, 37, 42, 58, 60, 61, 64, 65, 97, 103,
104, 105
- descriptors 1, 15
- DMSO 3, 9, 10, 111
- e-mail 10
- environment iii, 5, 12, 13, 18, 19, 21, 32
- Executive Agents iii
- extensions 1, 2, 13, 15, 30, 82, 97, 98
- Fortran 6, 15, 37, 103, 104
- FTP 16, 18, 19, 59, 94, 107, 111
- GrADS 17, 20, 24, 25, 26, 27, 29, 111
- grib_seek 8, 9, 31, 32, 34, 38, 40, 70, 94, 107
- gribsimp1, 12, 13, 14, 17, 18, 19, 20, 21, 22, 23, 24,
25, 27, 30, 95, 100, 103, 104, 105
- Grid Description Section 38, 45, 68, 82, 84, 107, 108,
111
- gunzip 11
- header 1, 15, 16, 18, 19, 20, 22, 32, 33, 34, 35, 36, 37,
38, 40, 44, 51, 67, 77, 78, 79, 88, 104, 108
- IEEE5, 6, 12, 17, 19, 24, 31, 37, 39, 43, 44, 45, 46,
47, 48, 49, 50, 56, 73, 93, 99, 100, 101, 103, 104,
105, 112
- IEEE files 5, 6, 12, 17, 19, 24, 31, 37, 39, 43, 44, 45,
46, 47, 48, 49, 50, 56, 73, 93, 99, 100, 101, 103,
104, 105, 112
- init_gribhdr 6, 8, 31, 32, 33, 38, 46, 49, 54, 67, 78, 88,
94, 108
- IRIX 13
- Lambert conformal grids 16, 20, 21, 108
- level definitions 2, 18, 19
- MEL Home page 9
- Modeling and Simulation iii, 111, 112
- ocean iii
- output 16, 88
- parameter definitions 60, 64, 74, 98
- PC users 11, 12
- Product Definition Section 15, 31, 32, 35, 37, 38, 58,
63, 64, 68, 69, 79, 80, 88, 97, 98, 107, 108, 112
- SGI 5, 13, 104, 112
- SGI and SGI IRIX 5, 13, 104, 112
- Software Library iii, x, 1, 5, 9, 11, 13, 14, 15, 16, 18,
31, 58, 59, 81, 84, 89, 93, 103, 107
- space iii
- spherical conformal grid 20, 21, 44, 85, 86, 87
- spherical conformal grids 16, 44, 83
- spherical harmonic coefficients 16
- Spherical Harmonic coefficients 16
- standard 1, 2, 14, 15, 17, 18, 22, 35, 42, 58, 65, 66,
78, 82, 83, 84, 97, 98
- Sun and SunOS 5
- SunOS 13, 104
- tar 11, 14, 17, 18, 22, 23, 24, 25, 27, 28, 30, 31, 33,
35, 93, 99, 112
- terrain iii
- UNIX 5, 18, 19, 99, 100
- Vis5D 13, 17, 21, 30
- WMO GRIB standard 2, 82, 97
- World Meteorological Organization (WMO) 1, 3, 112