METLIB utility package modified for new Fortran

Version 4.0

Bo Sundman, August 23, 2019

METLIB is a utility package originally written in Fortran 77[1] but has now been adapted to the new Fortran standard as a module. Most of the depreciated features has been removed and the "implicit none" declaration has been added to all the routines and the module as whole. It contains some routines that are now obsolete like concatenation of character variables and there may still be problems with features that will be depreciated as the new Fortran standard develops, please indicate any problems to the OC development team: support@pencalphad.com.

It contains calls to two C routines:

- The first is "getkex" which is a routine to read keyboard input character by character to allow command line editing. The original routine "getkey" was developed by John S. Urban in 2009.
- The other C routine is tinyfiledialogs to have a browser to open files for read and write. It was developed by Guillaume Vareille in 2014-2018.

There are some compiler switches available:

- -Dlixed include the getkex routine for command editing. It is not needed on Windows as Windows provides command line editing.
- -Dtinyfd include the tinyfiledialog as file browser.
- -Dlixhlp activate the online help on Linux and Mac OS. On Linux it also select the firefox browser.
- -Dmachlp specify the browser 0n Mac OS.

  On Windows the Explorer browser is used by default. The online help uses the "hypertarget" feature of HTML and LaTeX to find the relevant help text.

Contents

1 Introduction

This is a utility packge originally written in F77 and now converted to the new Fortran. It is a separate module and remaining depreciated features will be removed as soon as possible.

2 User interface

Most of the routines here deal with user interaction, asking for input, decoding commands and providing online help. For the online help the **hypertarget** feature in HTML is now used exclusively. The User Guide contains such hypertargets and whenever a question is asked the routine has such a target as argument and if the user types a ? the target is used as answer to a question. The help is provided in a browser window where the text in the User Guide is positioned at the place indicated by the hypertarget for the question, see section 4.5.2.

For help at menu commands the menu is provided if the user types ? but if the user types ?? extended help by the browser is provided.

2.1 Error handling

Handling input errors is one of the main difficulties in interactive programming. The routines in this package tries to detect such errors as early as possible and raise an error flag with an error code. This error code (0=no error) must be tested after each subroutine or function call and it is the responsibility of the calling routine to take the appropriate action.

At present the error code in the metlib package is not fully integrated in the OC error handling.

3 Data structures

Some of the global data structures which is presently declared in the model package "gtp", for example the error handling, will be moved to this package when convenient.

There are a few data structures inside the metlib package. These are for the symbol manipulations, the "PUTFUN" package, to allow defining expressions of state variables like the heat capacity "H.T" and for history and online help.

3.1 Data structure for PUTFUN

PUTFUN is a utility to store Fortran like expressions like a binary tree ad refer to it as a symbol. In this expression one can use other symbols or external variables such as state variables, the values of which are provided when calculating the symbols.

```
! Data structures in METLIB
  TYPE putfun_node
! all nodes of function stored as part of a binary tree
! kod is operation kod (0 datanod), links is how many links to this node
     integer kod,links
! this is the sequential order the node is allocated (for debugging)
     integer debug
! each node has a left and a right link.  If the left node is empty the
! right is normally a data node
     TYPE(putfun_node), pointer :: left,right
! A data node can have a  numeric value and/or a link to another function
     double precision value
! this is an identification of external symbols
     integer dataid
  end TYPE putfun_node
!
! BEWARE entering putfuns cannot be made in parallel processing
! but one may evaluate them in different threads
!
! PUTFUNNVAR is associated with external symbols in the LOKV array
  integer, private :: putfunvar
  TYPE PUTFUN_STACK
     type(putfun_node), pointer ::savetop, savebin, saveuni
     type(putfun_stack), pointer :: previous
  end TYPE PUTFUN_STACK
  type(putfun_stack), pointer :: stacktop
! topnod is the current top node
! lastopnod is last binary opkod node
! datanod is last data node
  TYPE(putfun_node), private, pointer :: topnod,datanod,lastopnod
  integer pfnerr,debuginc
!
! end data structures for PUTFUN
```

3.2 Data structures for history and online help

There are also data structures for the command history and to provide online help using a web browser.

```
! data structures for history and help
!
  integer, parameter :: histlines=100
```

```
!
  TYPE CHISTORY
! to save the last 20 lines of commands
     character*80 hline(histlines)
     integer :: hpos=0
  END TYPE CHISTORY
  type(chistory) :: myhistory
!
    integer, parameter :: maxhelplevel=15
! A help structure used in new on-line help system
! this was designed for both LaTeX and HTML help, now only HTML
    TYPE help_str
       integer :: okinit=0
       character*128 filename
       character*8 type
       integer level
       character*32, dimension(maxhelplevel) :: cpath
    END TYPE help_str
! this record is used to file the appropriate help text
    type(help_str), save :: helprec
! this is useful to add %\section and %\subsection in helpfile
    logical :: helptrace=.FALSE.
!
! using browser and html files for on-line help
  type onlinehelp
! if htmlhelp is TRUE then browser is the path/name of browser
! htmlfile is full path/name of html file
! target is used to find the relevant text the html file
! values of browser and htmlfile set by the main program (and htmlhelp=.TRUE.)
! The value of target is found searching the original LaTeX file!!
! In this file there are \hypertarget{target} which can be searched in the
! html file as <a id="target" />
! Searching the LaTeX file the help system will find a section
! matching the history of commands/questions the user has given
! and the target in the first \hypertarget {target} found within these lines
! will be used for the help displayed in the browser window
     logical :: htmlhelp=.FALSE.
     character*128 browser
     character*128 htmlfile
     character*128 latexfile
     character*64 target
  end type onlinehelp
  type(onlinehelp) :: ochelp
  save ochelp
! end data structures for history and help
```

3.3 System dependent variables

There are 3 system dependent constants needed for the WPACK routines defined in the beginning of METLIB:

- nwpr (Number of Words Per Real, default 2 as double precision is always used),
- nbitpw (Number of BITs per Word, default 32) and
- nbpw (Number of Bytes per Word, default 4).

```
! >>>>>>>>>> SYSTEM DEPENDENT <<<<<<<<<<
! nbpw is number if bytes per INTEGER, nwpr number of words per (double) real
! nbitpw number of bits per word
! USED when WPACK routines store data in integer workspace
    integer, parameter :: nbpw=4,nwpr=2,nbitpw=32
! >>>>>>>>>> SYSTEM DEPENDENT <<<<<<<<<<
```

4 Subroutines and functions

A complete list of subroutines and functions in alphabetical order (including the type of function) can be found in section 6. A rough arrangement of the routines belonging together has been made.

4.1 Sorting

There are 4 routines for sorting arrays of reals, double precision, integers and characters. 3 of them use the “quiksort” algorithm but the fourth is a simple “bubblesort” for characters with maximum length of 40 letters. None of them are used in any time-critical part of the OC software.

```
  SUBROUTINE SORTRD(ARR,N,IX)
! ...SORTING REAL NUMBERS IN ASCENDING ORDER
! INPUT:
!      ARR   ARRAY TO BE SORTED
!      N     NUMBER OF ELEMENTS TO BE SORTED >1
!      IX    INTEGER ARRAY WITH DIMENSION N
! EXIT:
!      ARR   SORTED ARRAY
!      IX    ARRAY WHERE IX(I) IS THE PREVIOS INDEX OF ARR(I)
    implicit none
    real ARR(*)
    integer n,ix(*)
---------------
  SUBROUTINE SORTRDD(ARR,N,IX)
! ...SORTING DOUBLE PRECISION NUMBERS IN DECENDING ORDER
! INPUT:
!      ARR   ARRAY TO BE SORTED
!      N     NUMBER OF ELEMENTS TO BE SORTED >1
!      IX    INTEGER ARRAY WITH DIMENSION N
! EXIT:
!      ARR   SORTED ARRAY
!      IX    ARRAY WHERE IX(I) IS THE PREVIOS INDEX OF ARR(I)
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    double precision ARR(*)
    integer n,ix(*)
---------------
  SUBROUTINE SORTIN(IARR,N,IX)
! ...SORTING INTEGERS IN ASCENDING ORDER
! INPUT:
!      IARR   ARRAY TO BE SORTED
!      N     NUMBER OF ELEMENTS TO BE SORTED >1
!      IX    INTEGER ARRAY WITH DIMENSION N
! EXIT:
!      IARR   SORTED ARRAY
!      IX    ARRAY WHERE IX(I) IS THE PREVIOS INDEX OF IARR(I)
    implicit none
    integer IARR(*),n,ix(*)
---------------
  SUBROUTINE SSORT(CMD,NS,INDEX)
!...SORTING a character array, max 40 characters long
    implicit none
    CHARACTER CMD(*)*(*)
    integer ns,index(*)
```

4.2 Routines to handle characters

Character manipulations is always a problem, these were originally designed for f77. Some of them can now be replaced by intrinsic routines.

4.2.1 Convert lower to upper case

```
  LOGICAL FUNCTION ucletter(ch1)
! returns TRUE if the character is A to Z
    implicit none
    character ch1*1
---------------
  CHARACTER FUNCTION BIGLET(CHA)
!...CONVERTS ONE CHARACTER FROM LOWER TO UPPER CASE
    implicit none
    CHARACTER*1 CHA
---------------
  SUBROUTINE capson(text)
! converts lower case ASCII a-z to upper case A-Z, no other changes
    implicit none
    character text*(*)
```

4.2.2 Scan for next non-blank character

This is very frequently used to skip spaces inside a character string (user input) to find the next non-blank letter. If all characters are spaces it returns TRUE, otherwise FALSE and the position if the non-blank character is indicated in the IP variable.

```
  LOGICAL FUNCTION EOLCH(STR,IP)
!...End of Line CHeck, TO SKIP SPACES FROM IP. RETURNS .TRUE. IF ONLY SPACES
!....MODIFIED TO SKIP TAB CHARACTERS ALSO
    implicit none
    CHARACTER STR*(*)
    integer ip
    integer, parameter :: ITAB=9
```

4.2.3 Extract a number from a character

Input is always read as a character string, these routines can extract a number from the character. Frequent use of the EOLCH routine to allow any number of blank characters between items.

```
  SUBROUTINE GETREL(SVAR,LAST,VALUE)
! extract a real from a character
    implicit none
    character svar*(*)
    integer last
    double precision value
---------------
  SUBROUTINE GETREM(SVAR,LAST,VAL)
! ...IDENTICAL TO GETREL EXCEPT THAT A TERMINATING COMMA "," IS SKIPPED
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER SVAR*(*)
    integer last
    double precision val
---------------
  SUBROUTINE GETRELS(SVAR,LAST,VALUE,ISIG)
!...DECODES A REAL NUMBER FROM A TEXT
!      IT MAY BE PRECEEDED BY SPACES AND A + OR -
!      THERE MUST BE AT LEAST ONE NUMBER BEFORE OR AFTER A PERIOD
!      THERE MUST BE AT LEAST ONE NUMBER BEFORE AN "E" OR "D"
!      AFTER AN "E" OR "D" THERE MAY BE A + OR - AND MUST BE ONE OR TWO NUMBERS
! 840310 CHANGE TO ALLOW SPACES AFTER A SIGN I.E. + 2.2 IS ALLOWED
! 860201 EXPONENTIAL D ACCEPTED
! 100910 F95 version
! ISIG is zero if no sign, needed to separte terms inside expressions
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    character svar*(*)
    integer last,isig
    double precision value
---------------
  INTEGER FUNCTION GPS(SVAR,LAST,VALUE)
!...DECODES A NUMBER WITH OR WITHOUT A SIGN
    implicit none
    DOUBLE PRECISION VALUE
    CHARACTER SVAR*(*)
    integer last
---------------
  INTEGER FUNCTION GPN(SVAR,LAST,VALUE)
!...DECODES A NUMBER WITHOUT SIGN
!    DOUBLE PRECISION VALUE
    implicit none
    CHARACTER SVAR*(*)
    integer last
    double precision value
```

4.2.4 Extract an integer, octal or hexadecimal number

Integer, octal or hexadecimal number are extracted from the character. The position in the character to start looking for the number is provided in the call.

```
  SUBROUTINE GETINT(SVAR,LAST,IVAL)
!...DECODES AN INTEGER FROM A TEXT
!      IT MAY BE PRECCEDED BY SPACES AND A + OR -
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER SVAR*(*)
    integer last,ival
---------------
  SUBROUTINE GETINM(SVAR,LAST,IVAL)
! ...IDENTICAL TO GETINT EXCEPT THAT A TERMINATING COMMA ",", IS SKIPPED
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER SVAR*(*)
    integer last,ival
---------------
  SUBROUTINE GETOCT(LINE,IP,IVAL)
!...DECODE AN OCTAL NUMBER
    implicit none
    CHARACTER LINE*(*)
    integer ip,ival
---------------
  SUBROUTINE GETHEX(LINE,IP,IVAL)
!...DECODE A HEXADECIMAL NUMBER
    implicit none
    CHARACTER LINE*(*)
    integer ip,ival
```

4.2.5 Extract a text or name from a character

Names of various items can be extracted from a character variable. A name should always start with a letter a-z, lower or upper case. This routine allows different ways of terminating the name. The position in the character to start looking for the name is provided in the call.

```
  subroutine getname(text,ip,name,mode,ch1)
! reading a species name, this should be incorporated in metlib,
    implicit none
    character text*(*),name*(*),ch1*1
    integer ip,mode
---------------
  SUBROUTINE GETEXT(SVAR,LAST,JTYP,STRING,CDEF,LENC)
!...SVAR SHALL CONTAIN A TEXT. SCAN STARTS AT POSITION LAST.
!      STRING IS SET TO THE FIRST NONBLANK CHARACTER UP TO THE TERMINATOR.
!      CDEF IS A DEFAULT VAUE IF SVAR IS EMPTY.
!      LENC IS THE LENGTH OF THE TEXT IN STRING
!      JTYP DEFINES THE TERMINATION OF A STRING
!      1 TEXT TERMINATED BY SPACE OR ","
!      2 TEXT TERMINATED BY SPACE
!      3 TEXT TERMINATED BY ";" OR "."
!      4 TEXT TERMINATED BY ";"
!      5 TEXT UP TO END-OF-LINE
!      6 TEXT UP TO AND INCLUDING ";"
!      7 text terminated by space but if first char is ', " up to next ' or "
!      8 text terminated by space but if first char is (, {, [ or < all text
!             until matching ), }, ] or >. Possibly including more ( ) etc.
!    >31, THE CHAR(JTYP) IS USED AS TERMINATING CHARACTER
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    CHARACTER SVAR*(*),CDEF*(*),STRING*(*)
    integer last,jtyp,lenc
```

4.3 Some routines for writing text

These routine are used to edit a real or integer number left justified into a character. There are also routines to write formatted output with a specified line length and left margin on a file.

```
  SUBROUTINE WRINUM(STR,IP,NNW,JSIGN,VALUE)
!...EDITS A REAL NUMBER INTO STR WITH LEAST NUMBER OF DIGITS
!      NNW IS MAXIMUM NUMBER OF SIGNIFICANT DIGITS (0<NNW<16)
!      JSIGN >0 INDICATES THAT + SIGN SHOULD BE WRITTEN
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER STR*(*)
    integer ip,nnw,jsign
    double precision value
---------------
  subroutine wriint(text,ipos,int)
! write an integer in text from position ipos (left adjusted)
    implicit none
    character text*(*),number*16
    integer ipos,int,jp
---------------
  SUBROUTINE WRIHEX(STR,IVAL)
!...TO WRITE AN INTEGER AS HEXADECIMAL
!    LOGICAL TESTB
    implicit none
    CHARACTER STR*(*)
    integer ival
---------------
  subroutine wrice(lut,margl1,margl2,maxl,str)
! writes str on unit lut with left margin largl1 for first line, margl2 for all
! following lines, max length maxl characters (assuming typewriter font)
    implicit none
    integer lut,margl1,margl2,maxl
    character str*(*)
---------------
```

```
  subroutine wrice2(lut,margl1,margl2,maxl,lbreak,str)
! writes str on unit lut with left margin largl1 for first line, margl2 for all
! following lines, max length maxl characters (assuming typewriter font)
! lbreak>0 for writing math expression, with stricter linebreak rules
! lbreak<0 for breaking only at space
    implicit none
    character str*(*)
    integer lut,margl1,margl2,maxl,lbreak
---------------
  subroutine cwricend(str,lbeg,lend,lbreak)
! find a possible place for a newline in str going back from lend
! but not bypassing lbeg.  str is a numerical expression.
! lbreak>0 means stricter rules (mathematical expression)
! lbreak<0 means break only at space
    implicit none
    character str*(*)
    integer lbeg,lend,lbreak
```

4.4 Command interpreter

These routines are used to interpret commands from the user. They are also connected to the history and online help routines to provide interactive help to a user. The command interpreter supports a MACRO facility to read commands from a file.

```
  INTEGER FUNCTION NCOMP(SVAR,COMM,NC,NEXT)
! SUBROUTINE NCOMP
    implicit none
    integer nc,next,ient
    CHARACTER SVAR*(*),COMM(NC)*(*)
---------------
  INTEGER FUNCTION NCOMP2(SVAR,COMM,NC,NEXT)
! SUBROUTINE NCOMP2
    implicit none
    integer nc,next,ient
    CHARACTER SVAR*(*),COMM(NC)*(*)
---------------
  INTEGER FUNCTION NCOMP3(SVAR,COMM,NC,NEXT)
! SUBROUTINE NCOMP3
    implicit none
    integer nc,next,ient
    CHARACTER SVAR*(*),COMM(NC)*(*)
---------------
  INTEGER FUNCTION NCOMPX(SVAR,COMM,NC,NEXT,IENT)
! ...TO DECODE A COMMAND
    implicit none
    CHARACTER SVAR*(*),COMM(NC)*(*)
    integer nc,next,ient
```

4.5 Prompt for command argument

After a command the user is normally asked for arguments of the command and the routines in this section either pick up arguments from the same input line as the command or it will prompt the user for the arguments.

If the user types ? or ?? as answer to a question the help routines will try to provide help either as part of the code or from the user guide.

4.5.1 Old routines to prompt for integer, real or character

These are depreciated prompt routines replaced by those in the next section.

```
  SUBROUTINE GPARID(PROMT,SVAR,LAST,IVAL,IDEF,HELP)
! ask for integer value with default
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER PROMT*(*),SVAR*(*)
    integer last,ival,idef
    EXTERNAL HELP
---------------
  SUBROUTINE GPARI_old(PROMT,SVAR,LAST,IVAL,IDEF,HELP)
! ask for integer value woth no default
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    CHARACTER PROMT*(*),SVAR*(*)
    integer last,ival,idef
    EXTERNAL HELP
---------------
  SUBROUTINE GPARR_old(PROMT,SVAR,LAST,VAL,RDEF,HELP)
! asks for a double with no default
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER PROMT*(*),SVAR*(*)
    integer last
    double precision val,rdef
    EXTERNAL HELP
---------------
  SUBROUTINE GPARRD_old(PROMT,SVAR,LAST,VAL,RDEF,HELP)
! ask for a double with default provided
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER PROMT*(*),SVAR*(*)
    integer last
    EXTERNAL HELP
    double precision val,rdef
---------------
  SUBROUTINE GPARC_old(PROMT,SVAR,LAST,JTYP,SVAL,CDEF,HELP)
! read a character without default
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    CHARACTER PROMT*(*),SVAR*(*),CDEF*(*),SVAL*(*)
    integer last,jtyp
    EXTERNAL HELP
---------------
  SUBROUTINE GPARCD_old(PROMT,SVAR,LAST,JTYP,SVAL,CDEF,HELP)
! read a character with default provided
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    CHARACTER PROMT*(*),SVAR*(*),CDEF*(*),SVAL*(*)
    integer last,jtyp
    EXTERNAL HELP
---------------
  subroutine GQARRD(PROMT,SVAR,LAST,VAL,RDEF,HELP)
! read real with default
    implicit none
    CHARACTER PROMT*(*),SVAR*(*)
    integer last,ival
    character*1 str,cdef
    double precision val,rdef
    EXTERNAL HELP
---------------
  subroutine GQARR(PROMT,SVAR,LAST,VAL,RDEF,HELP)
! read real without default
    implicit none
    CHARACTER PROMT*(*),SVAR*(*)
    integer last,ival
    EXTERNAL HELP
    double precision val,rdef
    character*1 str,cdef
---------------
  SUBROUTINE GQARID(PROMT,SVAR,LAST,IVAL,IDEF,HELP)
! previously subroutine GPARID
!...SVAR SHALL CONTAIN A PARAMETER VALUE. IF EMPTY THE PARAMETER IS ASKED FOR
!      USING PROMT AS OUTPUT STRING. IF NO ANSWER THE VALUE IN DEF IS RETURNED
!      INTEGER VALUES. THE DEFAULT VALUE IS DISPLAYED IN THE PROMT WITHIN
!      SLASHES. THE SAME ROUTINES WITHOUT THE FINAL D DOES NOT DISPALY THE
!      DEFAULT VALUE
!      HELP IS A ROUTINE THAT WRITES AN EXPLAINING MESSAGE.
!      LAST IS THE POSITION OF THE TERMINATOR OF THE FORMER PARAMETER OR
!      COMMAND, DECODING STARTS FROM THE POSITION AFTER LAST
    implicit none
    CHARACTER PROMT*(*),SVAR*(*)
    integer last,ival,idef
    character*1 str,cdef
    double precision val
    EXTERNAL HELP
---------------
  subroutine GQARI(PROMT,SVAR,LAST,IVAL,IDEF,HELP)
! read integer with no default
    implicit none
    CHARACTER PROMT*(*),SVAR*(*)
    integer last,ival,idef
    character*1 str,cdef
    double precision val
    EXTERNAL HELP
---------------
  subroutine GQARCD(PROMT,SVAR,LAST,JTYP,STR,CDEF,HELP)
! TO READ A STRING VALUE with default
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),str*(*),cdef*(*)
    integer last,jtyp
    EXTERNAL HELP
---------------
  SUBROUTINE gparall(PROMT,SVAR,LAST,IVAL,val,string,cdef,HELP)
! previously subroutine GPARID
!...SVAR SHALL CONTAIN A PARAMETER VALUE. IF EMPTY THE PARAMETER IS ASKED FOR
!      USING PROMT AS OUTPUT STRING. IF NO ANSWER THE VALUE IN DEF IS RETURNED
!      INTEGER VALUES. THE DEFAULT VALUE IS DISPLAYED IN THE PROMT WITHIN
!      SLASHES. THE SAME ROUTINES WITHOUT THE FINAL D DOES NOT DISPALY THE
!      DEFAULT VALUE
!      HELP IS A ROUTINE THAT WRITES AN EXPLAINING MESSAGE.
!      LAST IS THE POSITION OF THE TERMINATOR OF THE FORMER PARAMETER OR
!      COMMAND, DECODING STARTS FROM THE POSITION AFTER LAST
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),CH1*1,CDEF*(*),STRING*(*),SSD*30
    CHARACTER PPROMT*132,CH2*1
!    LOGICAL EOLCH,SG2ERR,WDEF,MATP
    LOGICAL WDEF,MATP
    EXTERNAL HELP
---------------
  subroutine GQARC(PROMT,SVAR,LAST,JTYP,STR,CDEF,HELP)
! read a string without default
```

```
    implicit none
    CHARACTER PROMT*(*),SVAR*(*)
    integer last,ival,jtyp
    EXTERNAL HELP
    double precision val
    character str*(*),cdef*(*)
---------------
    SUBROUTINE GPARFILE(PROMT,SVAR,LAST,JTYP,SVAL,CDEF,TYP,HELP)
! to ask for a file name using command line or external window
! prompt is question
! svar is a character variable which may already contain an answer
! last is position in svar to start searching for an answer
!      JTYP DEFINES THE TERMINATION OF A STRING
!      1 TEXT TERMINATED BY SPACE OR ","
!      2 TEXT TERMINATED BY SPACE
!      3 TEXT TERMINATED BY ";" OR "."
!      4 TEXT TERMINATED BY ";"
!      5 TEXT UP TO END-OF-LINE
!      6 TEXT UP TO AND INCLUDING ";"
!      7 TEXT TERMINATED BY SPACE OR "," BUT IGNORING SUCH INSIDE ( )
!    >31, THE CHAR(JTYP) IS USED AS TERMINATING CHARACTER
! sval is the answer either extracted from SVAR or obtained by user input
! cdef is a default answer
! typ  is default file extenion, at present only:
!  1=".TDB", 2=".UNF", 3=".OCM"
! help is a help routine
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    CHARACTER PROMT*(*),SVAR*(*),CDEF*(*),SVAL*(*)
    integer last,jtyp
    EXTERNAL HELP
```

4.5.2 Prompt for integer, real or character

These are a new almost identical set of prompt routines for command arguments using the new help feature. There are separate routines to ask for a character, integer or double precision real. The routines may provide a default value within slashes, /value/, which is accepted if the user press the enter/return key or a comma "," on the command line.

The position to read in the character is provided in the call and this will always be incremented by 1 to bypass the terminator of any previous argument. This simplifies asking several questions after each other.

```
  SUBROUTINE GPARIDx(PROMT,SVAR,LAST,IVAL,IDEF,hyper)
! ask for integer value with default
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),hyper*(*)
    integer last,ival,idef
!    EXTERNAL HELP
---------------
  SUBROUTINE GPARIx(PROMT,SVAR,LAST,IVAL,IDEF,hyper)
! ask for integer value woth no default
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),hyper*(*)
    integer last,ival,idef
!    EXTERNAL HELP
---------------
  SUBROUTINE GPARRx(PROMT,SVAR,LAST,VAL,RDEF,hyper)
! asks for a double with no default
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),hyper*(*)
    integer last
    double precision val,rdef
!    EXTERNAL HELP
---------------
  SUBROUTINE GPARRDx(PROMT,SVAR,LAST,VAL,RDEF,hyper)
! ask for a double with default provided
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),hyper*(*)
    integer last
!    EXTERNAL HELP
    double precision val,rdef
---------------
  SUBROUTINE GPARCDx(PROMT,SVAR,LAST,JTYP,SVAL,CDEF,hyper)
! read a character with default provided
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    CHARACTER PROMT*(*),SVAR*(*),CDEF*(*),SVAL*(*),hyper*(*)
    integer last,jtyp
    EXTERNAL HELP
---------------
  SUBROUTINE GPARCX(PROMT,SVAR,LAST,JTYP,SVAL,CDEF,hyper)
! read a character with default provided and hypertarget
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    CHARACTER PROMT*(*),SVAR*(*),CDEF*(*),SVAL*(*),hyper*(*)
    integer last,jtyp
!    EXTERNAL HELP now always use Q4HELP
```

```
---------------
  SUBROUTINE GQARIDX(PROMT,SVAR,LAST,IVAL,IDEF,hyper)
!...SVAR SHALL CONTAIN A PARAMETER VALUE. IF EMPTY THE PARAMETER IS ASKED FOR
!      USING PROMT AS OUTPUT STRING. IF NO ANSWER THE VALUE IN DEF IS RETURNED
!      INTEGER VALUES. THE DEFAULT VALUE IS DISPLAYED IN THE PROMT WITHIN
!      SLASHES. THE SAME ROUTINES WITHOUT THE FINAL D DOES NOT DISPALY THE
!      DEFAULT VALUE
!      hyper is a hypertarget for online help
!      LAST IS THE POSITION OF THE TERMINATOR OF THE FORMER PARAMETER OR
!      COMMAND, DECODING STARTS FROM THE POSITION AFTER LAST
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),hyper*(*)
    integer last,ival,idef
    character*1 str,cdef
    double precision val
---------------
  subroutine GQARIx(PROMT,SVAR,LAST,IVAL,IDEF,hyper)
! read integer with no default
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),hyper*(*)
    integer last,ival,idef
!    EXTERNAL HELP
---------------
  subroutine GQARRDx(PROMT,SVAR,LAST,VAL,RDEF,hyper)
! read real with default
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),hyper*(*)
    integer last
    double precision val,rdef
    EXTERNAL HELP
---------------
  subroutine GQARRx(PROMT,SVAR,LAST,VAL,RDEF,hyper)
! read real without default
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),hyper*(*)
    integer last
    double precision val,rdef
---------------
  subroutine GQARCDX(PROMT,SVAR,LAST,JTYP,STR,CDEF,hyper)
! TO READ A STRING VALUE with default
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),str*(*),cdef*(*),hyper*(*)
    integer last,jtyp
!    EXTERNAL HELP no longer needed
---------------
  subroutine GQARCX(PROMT,SVAR,LAST,JTYP,STR,CDEF,hyper)
```

```
! TO READ A STRING VALUE with default user hypertext
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),str*(*),cdef*(*),hyper*(*)
    integer last,jtyp
!    EXTERNAL HELP no longer needed
---------------
  SUBROUTINE gparallx(PROMT,SVAR,LAST,IVAL,val,string,cdef,hyper)
! this is the focal routine for all variants of GPARxyz
!...SVAR shall contain an answer or command. If EMPTY THE answer IS ASKED FOR
!      USING PROMT AS OUTPUT STRING. IF NO ANSWER THE VALUE IN DEF (default)
!      is returned if a provided.  The routine can return integer, double or
!      character variables. THE DEFAULT VALUE IS DISPLAYED IN THE PROMT WITHIN
!      SLASHES. if no answer and no defualt an error is returned.
!      HELP is no longer a parameter Q4HELP is always used
!      as hypertarget in a HTML file
!      If hyper contains the character ?TOPHLP and the user has typed a single ?
!      the routine returns with this ? and the calling routine can display
!      a menu.  If the user types two ?? the PROMT is used as hypertarget.
!      LAST IS THE current POSITION IN SVAR, it is incremented by one
!      before looking for an answer (skipping the terminator of any previous
!      input.
! REPEAT:
! when called on top level or from a submenu then hyper='?TOPHLP'
! if user types a single ? only menu listed, with ?? use PROMT as target
    implicit none
    CHARACTER PROMT*(*),SVAR*(*),CH1*1,CDEF*(*),STRING*(*),hyper*(*)
    integer ival
    double precision val
!    EXTERNAL HELP
```

4.5.3 Ask for a file name using file browser (or not)

This is a subroutine to open a file for read or write. If the tinyfiledialog package is used it will open a file browser to select directory and file. Otherwise the user must type a full or relative path to the file.

```
    SUBROUTINE GPARFILEx(PROMT,SVAR,LAST,JTYP,SVAL,CDEF,TYP,hyper)
! to ask for a file name using command line or external window
! prompt is question
! svar is a character variable which may already contain an answer
! last is position in svar to start searching for an answer
!      JTYP DEFINES THE TERMINATION OF A STRING (maybe redundant??)
!      1 TEXT TERMINATED BY SPACE OR ","
!      2 TEXT TERMINATED BY SPACE
!      3 TEXT TERMINATED BY ";" OR "."
!      4 TEXT TERMINATED BY ";"
!      5 TEXT UP TO END-OF-LINE
!      6 TEXT UP TO AND INCLUDING ";"
!      7 TEXT TERMINATED BY SPACE OR "," BUT IGNORING SUCH INSIDE ( )
!    >31, THE CHAR(JTYP) IS USED AS TERMINATING CHARACTER
! sval is the answer either extracted from SVAR or obtained by user input
! cdef is a default answer
! typ  is default file extenion, at present only:
!  1=".TDB", 2=".UNF", 3=".OCM"
! hyper is a hypertext target for help
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    CHARACTER PROMT*(*),SVAR*(*),CDEF*(*),SVAL*(*),hyper*(*)
    integer last,jtyp
!    EXTERNAL HELP
```

4.6 Online help

It is always difficult to provide help to interactive software. In this package all routines in section 4.5.2 to ask a question has an argument that is a help routine. This help routine may provide direct help or it may use the history facility to search for help in the user guide. The on line help uses the “hypertarget” feature implemented in HTML and LaTeX/PDF to seach the user guide for a relevant help. If this is found the section found in the user guide will de displayed in a separate browser window. In this window the user may scroll the whole user guide to find the help he requires.

This feature require constant updating of the user guide whenever there are changes in the software and it may often not be updated, in particular in pre-released versions of the software.

```
  subroutine init_help(browser,htmlfile)
! This routine is called from oc_command_monitor to inititate
! the on-line help system. It saves the name of the browser and HTML file
    implicit none
    character*(*) htmlfile,browser
    character*80 line
---------------
  subroutine helplevel1(line)
! This routine is called from the monitor for the top level command
! It initiates the path to find the correct help text
! In all gparx routines the help level in increased and the question saved
    implicit none
    character*(*) line
---------------
  subroutine q1help(prompt,line)
```

```
! This routine is called from all gparx routines
! when the user types a ?
! prompt is never used ...
    implicit none
    character*(*) prompt,line
    character hline*80,mtext*12
    integer, parameter :: maxlevel=20
---------------
  subroutine q2help(prompt,line)
! This routine is called from submenu
! when the user types a ?
    implicit none
    character*(*) prompt,line
---------------
  SUBROUTINE Q3HELP(LINE,LAST,COMM,NC)
! used in submeny when user gives "? 'command' " taken as "help 'command'"
!...EXECUTES A HELP COMMAND
    implicit none
    CHARACTER LINE*(*),COMM(NC)*(*)
    integer last
---------------
  SUBROUTINE Q3HELPx(LINE,LAST,COMM,NC)
! used in submeny when user gives "? 'command' " taken as "help 'command'"
!...EXECUTES A HELP COMMAND
    implicit none
    CHARACTER LINE*(*),COMM(NC)*(*)
    integer last
---------------
  subroutine q4help(hypertarget,extra)
! This routine is adapted to provide help from webrowsers using hypertarget
! when the user types a ? or ??
    implicit none
    integer extra
    character*(*) hypertarget
---------------
  SUBROUTINE NOHELP(PROMT,LINE)
! no help available
    implicit none
    CHARACTER PROMT*(*),LINE*(*)
---------------
  SUBROUTINE TOPHLP(PROMPT,LINE)
! return to calling routine for help, do not save the current command ...
    implicit none
    CHARACTER PROMPT*(*),LINE*(*)
---------------
  LOGICAL FUNCTION YESCHK(CH1)
```

```
!    returns TRUE if CH1 is Y or y
    CHARACTER CH1*1
```

4.7 Command history

This saves all commands typed by the user and allows listing and recalling of previous commands. If the command line editing feature is available, the getkex routine, see section 4.8, the user may edit a previous command before executing it again.

```
  SUBROUTINE NGHIST(LINE,LAST)
!...EXECUTES A HISTORY COMMAND
!       LAST IS SET TO 0 IF LINE IS SET TO A COMMAND FROM HISTORY LIST
!    CHARACTER HIST*80,LINE*(*),CH1*1
    implicit none
    CHARACTER LINE*(*)
    integer last
---------------
  subroutine openlogfile(name,text,lun)
! opens a logfile for commands
    implicit none
    character name*(*),text*(*)
    integer lun
---------------
  subroutine set_echo(ion)
! set echo of command input, does this really work?
    implicit none
    integer ion
```

4.8 Command prompt and command line input with editing

These write a command prompt or question and read input with possible line editing. On Linux and Mac OS the getkex routine is needed for command line editing.

```
  subroutine boutxt(lut,line)
! writes the text on line on unit lut without CR/LF
    implicit none
    integer lut
    character line*(*)
---------------
  subroutine bintxt(lin,cline)
! read a command line with or without arguments. On LINUX command line editing
    implicit none
    character cline*(*)
    integer lin
---------------
  subroutine bintxt_getkey(lin,cline)
! LINUX subroutine to read a line with history and editing a la emacs
!
    implicit none
    character cline*(*)
    integer lin
---------------
  subroutine bintxt_nogetkey(lin,line)
! Reading a command line on Windows with editing provided by the OS
    implicit none
    character line*(*)
    integer lin
```

4.9 Command macro routines

The user can prepare a sequence of command on a text file and execute them with a “macro <filename> command. A macro can call another macro 5 levels deep. When a macro terminates the calling macro or the user command level is restored.

A macro can contain variables that are set by by the user when running the macro. There are no loops or conditional jumps.

```
  SUBROUTINE MACBEG(LINE,LAST,OK)
!....subroutine to execute set-interactive allowing nesting of macros
!
! IDEA: addera lablar i macro sa man kan ange MACRO fil LABEL
! och vid stop som @? eller @& man kan interaktivt ange GOTO label
! Ocksa en generisk subrutin som gor att man kan fa fram ett variabelvarde
! call macsymval(package,symbol,ival,rval,cval)
!
    implicit none
    CHARACTER LINE*(*)
    LOGICAL OK
    integer last
---------------
    SUBROUTINE MACEND(LINE,LAST,OK)
! end of macro detected, close file and return to upper level
      implicit none
      CHARACTER LINE*(*)
      LOGICAL OK
      integer last
---------------
  SUBROUTINE GPTCM1(IFLAG,SVAR,LAST,SLIN)
```

```
!...handling of MACRO directives like @& @? and @# etc
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER SVAR*(*),slin*(*)
    integer iflag,last
---------------
  subroutine GPTCM2(IFLAG,SVAR,LAST,SLIN)
! handling of macro variables
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER SVAR*(*),slin*(*)
    integer iflag,last
---------------
  SUBROUTINE GQXENV(SVAR)
!...EXCHANGES REFERENCES TO ENVIRONMENT MACRO VARIABLES TO ACTUAL VALUES
!       REFERENCES ARE FOR EXAMPLE ##4
!    CHARACTER SVAR*(*),ENVIR(*)*(*),CH1*1,LABLIN*60,LABEL*8
    implicit none
    CHARACTER SVAR*(*)
```

4.10 Routines to create, calculate and list a function as a binary tree

The PUTFUN library create symbols with Fortran like expression including some functions like LOG, LN, EXP etc using state variables or other symbols defined in OC. The expression is stored as a binary tree and can be calculated whenever the user demands.

Note it calculates a single value, the TP function package in the "models" package of OC calculates also the first and second derivatives of a parameter function with respect to T and P. Thus the TP function package has a much more restricted syntax.

This is the main routine to enter an expression as a character string.

```
  SUBROUTINE PUTFUN(STRING,L,MAXS,SYMBOL,LOKV,LROT,ALLOWCH,NV)
!...READS AN EXPRESSION FROM STRING POSITION L AND CREATES AN BINARY TREE
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER STRING*(*),SYMBOL(*)*(*)
    integer LOKV(*)
    integer maxs,allowch,nv
!    type(putfun_symlink) :: symlist
    LOGICAL NOTPP
    TYPE(putfun_node), pointer :: LROT
```

4.10.1 Routines to compile the expression

These routines are used to compile the expression into a binary tree.

```
  SUBROUTINE NYBIN(kod,binnod,NOTPP)
!...INSERTS A NEW OPNODE IN THE TREE
    implicit none
    integer kod
    TYPE(putfun_node), pointer :: binnod
    LOGICAL NOTPP
---------------
  SUBROUTINE NYUNI(KOD,negmark,uninod,IPN,NOTPP)
!   Creates a node with a unary operator
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    TYPE(putfun_node), pointer :: UNINOD
    LOGICAL NOTPP
    integer kod,negmark,ipn
---------------
  SUBROUTINE NYLP(uninod,IPN,NOTPP)
!...OPENING PARENTHESIS, push links on LEVEL. Also after unary operator
    implicit none
    TYPE(putfun_node), pointer :: uninod
    integer ipn
    LOGICAL NOTPP
---------------
  subroutine NYRP(IPN,NOTPP)
!...CLOSING PARENTHESIS
!    implicit double precision (a-h,o-z)
    implicit none
    integer ipn
    LOGICAL NOTPP
---------------
  SUBROUTINE NYVAR(TEXT,L,IOPUNI,negmark,MAXS,SYMBOL,LOKV,allowch,dummy2)
! inserts a symbol in an expression
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER TEXT*(*),SYMBOL(*)*(*)
    integer LOKV(*)
    integer iopuni,negmark,maxs,allowch
    type(putfun_node), pointer :: dummy2
---------------
  SUBROUTINE NYDAT(KOD,VAL,nynod,negmark)
! store a constant or symbol.  The address to the node is returned in lok
! which is used if the symbol is used several times.
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
```

```
    implicit none
    integer kod,negmark
    TYPE(putfun_node), pointer :: nynod
    double precision val
```

4.10.2 Routines to calculate the expression

These routines are used to calculate the value of the expression.

```
  double precision function evalf(LROT,VAR)
!      Calculates the value of an expression MEMORY LEAK
! ?? I do not know what is the difference with evalf_x ??/BoS 190804
!
! VAR is array with values of symbols that can be referenced
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    double precision VAR(*)
    type(putfun_node), pointer :: lrot
---------------
  double precision FUNCTION EVALF_X(LROT,VAR)
!      Calculates the value of an expression
! ?? I do not know what is the difference with evalf ??/BoS 190804
!
! VAR is array with values of symbols that can be referenced
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    type(putfun_node), pointer :: lrot
    double precision VAR(*)
---------------
  SUBROUTINE EUNARY(KOD,X)
! calculates a unary function such as LOG, EXP etc
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    integer kod
    double precision X
---------------
  SUBROUTINE EBINRY(KOD,X,Y)
! Calculates the value of a binary node with two data nodes
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    integer kod
    double precision X,Y
---------------
  double precision FUNCTION AIVAN(PECN)
!      CALCULATES THE DIMENSIONLESS SUPERCOOLING OF DIFFUSION BY
!      IVANTSOV'S SOLUTION
!...added by Zikui and also an updated ERF
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    double precision PECN
!      APPROXIMATIVE FORMULA FOR ERROR FUNCTION GIVEN BY:
!      ABRAMOWITZ AND STEGUN: HANDBOOK OF MATHEMATICAL FUNCTIONS,
!      NATIONAL BUREAU OF STANDARDS, 9TH EDITION, 1970
---------------
  double precision FUNCTION PF_BSUM(FA)
!.. 1993-10-06 20:10:56 /BJ
!
!                                 ( sin(n*pi*f) )^2
!  Calc. the infinit sum B = sum(-------------------)
!                                    (n*pi)^3
!
!.. If we truncate the sum at N=200 the relative error is
!   less than ?% for 0.01 < F < 0.99
!
!
    implicit none
!    IMPLICIT DOUBLE PRECISION(A-H,O-Z)
    double precision FA
---------------
  double precision FUNCTION PF_HS(X)
!      Calculates Heaviside function
!    IMPLICIT DOUBLE PRECISION(A-H,O-Z)
    implicit none
    double precision X
---------------
  double precision FUNCTION PF_ERF(X0)
!      CALCULATES ERROR-FUNCTION OF X, USING AN
!      APPROXIMATIVE FORMULA GIVEN BY:
!      ABRAMOWITZ AND STEGUN: HANDBOOK OF MATHEMATICAL FUNCTIONS,
!      NATIONAL BUREAU OF STANDARDS, 9TH EDITION, 1970
    implicit none
    double precision X0
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
```

4.10.3 Routines to list the expression

These routines are used to list the expression as a character string. This can then be written on an output unit.

```
  SUBROUTINE WRTFUN(STRING,IPOS,LROT,SYMBOL)
```

```
!      Writes a PUTFUN expression
!
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER STRING*(*),SYMBOL(*)*(*)
    integer ipos
    type(putfun_node), pointer :: lrot,current,lnode,rnode,tnode
---------------
  SUBROUTINE WRTLPQ(STRING,IPOS,LINK,KOD,LOD,negmark)
! write a left ( or unary operator followed by (
! the unary operator is in LOD
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER STRING*(*)
    integer ipos,link,kod,lod,negmark
---------------
  SUBROUTINE WRTRPQ(STRING,IPOS,LINK,KOD,LOD)
!  write a right )  but if LOD<-1 do not write (
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER STRING*(*)
    integer ipos,link,kod,lod
---------------
  SUBROUTINE WRTBIQ(STRING,IPOS,KOD)
! write a binary operator
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    CHARACTER STRING*(*)
    integer ipos,kod
---------------
  SUBROUTINE WRTDAQ(STRING,IPOS,KOD,VAL,SYMBOL,negmark)
!     write a number, if KOD<0 a whole number
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    CHARACTER NAME*8,SYMBOL(*)*(*)
    CHARACTER STRING*(*)
    integer ipos,kod,negmark
    double precision val
---------------
  SUBROUTINE CONS(STR1,IPOS,STR2)
! used in PUTFUN but should be replaced by //
    implicit none
    CHARACTER STR1*(*),STR2*(*)
    integer ipos
```

4.10.4 Routines to enter interactively or delete an expression

These routines are used to enter the expression and to delete it.

```
  SUBROUTINE EXPHLP(PROMPT,SVAR)
! writes help to enter a PUTFUN expression
    implicit none
    CHARACTER PROMPT*(*),SVAR*(*)
---------------
  SUBROUTINE PUTPRP(NAMN,MAXS,SYMBOL,PROMPT,ILEN)
!...CREATES A PROMPT asking for a putfun expression with formal arguments
    implicit none
    CHARACTER NAMN*(*),PROMPT*(*),SYMBOL(*)*(*)
    integer ilen,maxs
!...write a prompt with name of all variables
---------------
  SUBROUTINE DELFUN(LROT,IWS)
!   delete a putfun expression :: not converted to structures
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    integer IWS(*)
    integer lrot
```

4.11 HP calculator

This ia a small interactive online calculator using HP inverted polish notation.

```
  SUBROUTINE HPCALC
!...EMULATES A HP CALCULATOR ON SCREEN
    implicit none
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
---------------
  SUBROUTINE HPHLP
! writes a help text for using the online HP calculator
    implicit none
```

4.12 WPACK routines for unformatted save/read

These routines are used to store characters, integers and double precision reals into an integer "workspace". This workspace can be written on an "unformatted" Fortran file and read back to restore all data saved.

Using the new Fortran TYPE variables all these must be converted separately and the “pointers” replaced by the integer indices where the variable is stored. It is a very useful but fragile feature.

4.12.1 Initiate, write and read a workspace

A workspace is an integer array with the dimenstion NWT. When the workspace is initiated a certain numer of words, NWR, are reserved and zeroed, the rest of the workspace can be reserved dynamically. The first word of the worspace is the index, “pointer” to the first free area in the to dynamic area available to reserve. The second word is the size of the workspace.

From word 3 until the end of the initially reserved area the words are used to store “pointers”, i.e. the index of records in the dynamic area.

Inside the dynamic workspace a free list is maintained by the routines for reserving and releasing records. The first word of a free part of the workspace is the index, “pointer”, to the next free part of the workspace. In the last free workspace this “pointer” is zero. The second word in a free part of the dynamic workspace is the total number of free words in this area up to the next reserved record. A free space must thus be at least two words.

When saving on a file the first word is the number of words written on the file, then the workspace is written as one block. When reading this file back into a workspace the first word on the file is used to determine the number of words to read. These are directly copied into the new workspace. If the workspace for reading is larger than that used for saving the content of word 2 and in the last free workspace is updated.

```
  SUBROUTINE WINIT(NWT,NWR,IWS)
!...INITIATES A WORKSPACE
! INPUT: NWT IS THE DIMENSION OF THE WORKSPACE
!        NWR IS THE NUMBER OF WORDS TO BE EXCLUDED IN THE BEGINNING
!        IWS IS THE WORKSPACE
! EXIT:  THE FREE LIST IS INITIATED IN IWS
! ERRORS: NWR LESSER THAN ZERO
!         NWT LESSER THAN NWR+100
    implicit none
    integer nwt,nwr,iws(*)
!    DIMENSION IWS(*)
---------------
  SUBROUTINE WOLD(FIL,NW,IWS)
!...READS A FILE INTO A WORKSPACE. THE FILE MUST HAVE BEEN WRITTEN BY WSAVE
! INPUT: FIL A CHARACTER WITH A LEGAL FILE NAME
!        NW THE DIMENSION OF IWS
!        IWS THE WORKSPACE
! CALLS: WRKCHK TO CHECK THE FREE LIST
! EXIT:  THE CONTENT OF THE FILE IS IN IWS. THE DIMENSION OF IWS IS SET TO
!            NW AND THE LAST FREE AREA IS CORRECTED
    implicit none
    CHARACTER FIL*(*)
    integer nw,iws(*)
!    DIMENSION IWS(*)
---------------
  SUBROUTINE WSAVE(FIL,NW,IWS)
!...WRITES A WORKSPACE ON A FILE
! INPUT: FIL IS A CHARACTER WITH A LEGAL FILE NAME
!        NW IS THE DIMENSION OF THE WORKSPACE
!        IWS IS THE WORKSPACE
! CALLS: WRKCHK TO CHECK THE WORKSPACE
! ERROR: IF THE WORKSPACE IS INCORRECT IT CANNOT BE SAVED
    implicit none
    integer nw,iws(*)
!    DIMENSION IWS(*)
    CHARACTER FIL*(*)
```

4.12.2 Listing and interactive patching the workspace

These routines are mainly obsolete now.

```
  SUBROUTINE WPATCH(NW,IWS)
!...ROUTINE TO PATCH A WORKSPACE
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    integer nw,iws(*)
!    DIMENSION IWS(*)
---------------
  SUBROUTINE WPHLP(ITYP,LINE)
!...HELP ROUTINE FOR WPATCH
    implicit none
    CHARACTER LINE*(*)
    integer ityp
---------------
  SUBROUTINE WRKCHK(LAST,NW,IWS)
!...CHECKS THE FREE LIST IN A WORKSPACE
! INPUT: NW IS THE DIMENSION
!        IWS IS THE WORKSPACE
! EXIT:  LAST IS PUT TO THE LAST WORD USED IN THE WORKSPACE
! ERRORS: ANY ERROR IN THE FREE LIST (POINTER OUTSIDE WORKSPACE ETC)
    implicit none
    integer last,nw,iws(*)
!    DIMENSION IWS(*)
```

```
---------------
  SUBROUTINE WLIST(IWS)
!...LISTS THE FREE AREAS
    implicit none
    integer iws(*)
!    DIMENSION IWS(*)
```

4.12.3 Reserving and releasing records in the workspace

These routines reserve and release records in the dynamic workspace. The index of the first reserved word is a “pointer” to the record.

The WTREST routine reserves the remaining part of a workspace and initiates a free list in this for use by some other software. The WTAKE routine is called with the number of words needed for the record. The free list is searched for a free area which fits the request exactly or is at least 2 words longer (needed for the free list). The index of the first word of the reserved record is returned as “pointer” and all words reserved are set to zero. The WRELS routine is called with the location of a record to be released and how many words that should be released. It searches the free list for a free block just before the record to be released. It checks if the released record can be merged with the preceeding free area or a free area following the released area. It updates the pointers and sizes of the free areas.

```
  SUBROUTINE WTREST(NYB,NW,IWS)
!...RESERVES THE LAST PART OF THE WORKSPACE
! INPUT: IWS IS A WORKSPACE
! EXIT:  NYB IS A POINTER TO THE RESERVED PART
!        NW IS THE NUMBER OF RESERVED WORDS
    implicit none
    integer nyb,nw,iws(*)
!    DIMENSION IWS(*)
---------------
  SUBROUTINE WTAKE(NYB,NW,IWS)
!......RESERVS NW WORDS IN THE WORKSPACE
! INPUT: NW IS THE NUMBER OF WORDS TO BE RESERVED
!        IWS IS THE WORKSPACE
! EXIT:  NYB POINTS TO THE FIRST WORD THAT IS RESERVED
! ERROR: TOO SMALL OR TOO LARGE NUMBER OF WORDS TO BE RESERVED
    implicit none
    integer nyb,nw,iws(*)
!    DIMENSION IWS(*)
!...THE FREE LIST START IN THE FIRST WORD
!      IN EACH FREE AREA THE FIRST WORD POINTS TO THE NEXT FREE AREA
!      AND THE SECOND GIVES THE NUMBER OF WORDS IN THIS AREA
!      THE FREE LIST ENDS WITH THE POINTER EQUAL TO ZERO
---------------
```

```
  SUBROUTINE WRELS(IDP,NW,IWS)
!......Returns NW words beginning from IDP to the free workspace list
!      The free workspace list is in increasing order
!      IWS(1) points to the first free space
!      IWS(2) gives the total number of words in the workspace
    implicit none
!    DIMENSION IWS(*)
    integer idp,nw,iws(*)
!......Check that the released space is at lest 2 words and that it is
!      inside the workspace (That is between 3 and IWS(2))
```

4.12.4 Storing characters and doubles in the workspace

These routines copy characters and double precision reals in the workspace. Integer values are stored as they are.

The routines for storing characters and double precision reals copies the exact bit pattern of the original data to and from the integer workspace. This may be a bit clumsy but that is due to the fact it must bypass some checks made by the compiler of subroutine arguments. In the OC package the writing/reading of the workspace is normally not a time critical part of the code.

Funny things can happen transfering characters and reals to integers, in 1980 when this workspace package was first tested we initially stored 2 bytes per work using a Hollerith "H2" format which worked well on our Nord-10. But when the code was later compiled on a PDP-11 the order of the bytes were switched so a text as "HELLO WORD" came back as "EHLL OOWDR". Then we realized we had to transfer the bit pattern for the whole variable.

```
  INTEGER FUNCTION NWCH(NB)
! number of words to store a character with nb bytes
! nbpw is the number of bytes in a word.  If not even multiple add 1 word
    implicit none
    integer nb
---------------
  SUBROUTINE STORC(N,IWS,text)
! Stores a text character in an integer workspace at position N
! The length of the character to store is len(text)
---------------
  SUBROUTINE LOADC(N,IWS,text)
! copies a text from an integer workspace at position N into a character
! The number of characters to copy is len(text)
    implicit none
    integer n,iws(*)
    character text*(*)
```

```
!    character (len=:), allocatable :: localtxt
!    integer, allocatable, dimension(:) :: localint
! maximal size of character, note used also to store functions and bibliography
---------------
  SUBROUTINE STORR(N,IWS,VALUE)
!...STORES A REAL NUMBER IN A WORKSPACE at index N
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
!    DIMENSION IWS(*)
    implicit none
    integer iws(*)
    double precision value
    integer n
---------------
  SUBROUTINE LOADR(N,IWS,VALUE)
!...LOADS A REAL NUMBER FROM A WORKSPACE at index N
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    integer iws(*)
!    DIMENSION IWS(*)
    DOUBLE PRECISION VALUE
    integer N
---------------
  SUBROUTINE STORRN(N,IWS,ARR)
! store N doubles in workspace
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
!    DIMENSION IWS(*),ARR(*)
    integer n,iws(*)
    double precision arr(*)
---------------
  SUBROUTINE LOADRN(N,IWS,ARR)
! load N doubles from workspace
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    double precision ARR(*)
    integer n,iws(*)
    integer, parameter :: maxr=256
    double precision dlocal(maxr)
---------------
  SUBROUTINE STORR1(ARR,VAL)
! store a single double in workspace
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    double precision arr,val
---------------
  SUBROUTINE LOADR1(ARR,VAL)
```

```
! load a single double from workspace
!    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    implicit none
    double precision arr,val
```

4.13 Indexing a 2D array

In the OC package 2D arrays are frequently used to store results of symmetric second derivatives. As only the upper half of such matrices are needed the functions below calculate the sequential storage position. However, running large OC calculations show these routines may take up to 10% of the execution time and an attempt has been made to speed this indexing up using the kxsym routine.

```
  integer function ixsym(ix1,ix2)
! calculates the storage place of value at (i,j) for a symmetrix matrix
! storage order 11, 12, 22, 13, 23, 33, etc
    implicit none
    integer ix1,ix2
---------------
  integer function kxsym(ix1,ix2)
! calculates the storage place of value at (i,j) for a symmetrix matrix
! storage order 11, 12, 22, 13, 23, 33, etc
! In OC the calls to ixsym take about 10 % of the CPU time
! I am trying to replace with local indexing but I need a routine
! that calculates the index when both indices are equal or when I know
! the second index is larger
!    if(ix1.le.0 .or. ix2.le.0) then
!       buperr=1000; goto 1000
!    endif
    implicit none
    integer ix1,ix2
```

4.14 Miscellaneous

These are routines that does not fit in any of the sections above.

```
  subroutine fxdflt(file,ext)
! add default file extention, no good as it thinks .. is an externtion
    implicit none
    character file*(*),ext*(*)
---------------
  subroutine iniio
! initiates i/o variables, they are all global variables
```

```
    implicit none
---------------
  SUBROUTINE FISEPA(STR,IP0,IP1)
!...FINDS A SEPARATOR AFTER POSITION IP0
!      A separator is:
!      Any character exept A-Z, 0-9 and _
    implicit none
    CHARACTER STR*(*)
    integer IP0,IP1
---------------
  SUBROUTINE FDMTP(LINE1,IP,LINE2)
!...FINDS A MATCHING ) AFTER THAT AT IP. IP UPDATED TO POSITION AFTER )
    implicit none
    CHARACTER LINE1*(*),LINE2*(*)
    integer ip
---------------
  INTEGER FUNCTION KNDEX(LINE,IP,SS)
! SUBROUTINE KNDEX
!...SEARCHES FOR STRING SS IN LINE FROM IP
    implicit none
    CHARACTER LINE*(*),SS*(*)
    integer ip
---------------
  SUBROUTINE CPSSTR(STRING,LC)
!...THIS SUBROUINE COMPRESSES STRING BY REPLACING MULTIPLE SPACES
! OR TABS WITH A SINGLE SPACE
    implicit none
    CHARACTER STRING*(*)
    integer LC
---------------
  SUBROUTINE UNTAB(LINE)
!...REMOVES ALL TABS FROM LINE. INSERTS SPACES UP TO NEXT TAB STOP
!       TAB STOPS GIVEN IN ITABS. TABS AFTER POSITION 80 REPLACED
!       WITH A SPACE
    implicit none
    CHARACTER LINE*(*)
```

5 Summary

That is all!

6 List of all subroutines and functions

Tables with 133 functions and subroutines

Name	File
character function biglet	metlib4.F90
double precision evalf	metlib4.F90
double precision function aivan	metlib4.F90
double precision function evalf_x	metlib4.F90
double precision function pf_bsum	metlib4.F90
double precision function pf_erf	metlib4.F90
double precision function pf_hs	metlib4.F90
integer function gpn	metlib4.F90
integer function gps	metlib4.F90
integer function ixsym	metlib4.F90
integer function kndex	metlib4.F90
integer function kxsym	metlib4.F90
integer function ncomp	metlib4.F90
integer function ncomp2	metlib4.F90
integer function ncomp3	metlib4.F90
integer function ncompx	metlib4.F90
integer function nwch	metlib4.F90
logical function eolch	metlib4.F90
logical function ucletter	metlib4.F90
logical function yeschk	metlib4.F90
subroutine bintxt	metlib4.F90
subroutine bintxt_getkey	metlib4.F90
subroutine bintxt_nogetkey	metlib4.F90
subroutine boutxt	metlib4.F90
subroutine capson	metlib4.F90
subroutine cons	metlib4.F90
subroutine cpsstr	metlib4.F90
subroutine cwicend	metlib4.F90
subroutine delfun	metlib4.F90
subroutine ebinary	metlib4.F90
subroutine eunary	metlib4.F90
subroutine exphlp	metlib4.F90
subroutine fdmtp	metlib4.F90
subroutine fisepa	metlib4.F90
subroutine fxdflt	metlib4.F90
subroutine getext	metlib4.F90
subroutine gethex	metlib4.F90
subroutine getinm	metlib4.F90

Name	File
subroutine getint	metlib4.F90
subroutine getname	metlib4.F90
subroutine getoct	metlib4.F90
subroutine getrel	metlib4.F90
subroutine getrels	metlib4.F90
subroutine getrem	metlib4.F90
subroutine gparall	metlib4.F90
subroutine gparallx	metlib4.F90
subroutine gparc	metlib4.F90
subroutine gparcd	metlib4.F90
subroutine gparcdx	metlib4.F90
subroutine gparcx	metlib4.F90
subroutine gparfile	metlib4.F90
subroutine gparfilex	metlib4.F90
subroutine gpari	metlib4.F90
subroutine gparid	metlib4.F90
subroutine gparidx	metlib4.F90
subroutine gparix	metlib4.F90
subroutine gparr	metlib4.F90
subroutine gparrd	metlib4.F90
subroutine gparrdx	metlib4.F90
subroutine gparrx	metlib4.F90
subroutine gptcm1	metlib4.F90
subroutine gptcm2	metlib4.F90
subroutine gqarc	metlib4.F90
subroutine gqarcd	metlib4.F90
subroutine gqarcdx	metlib4.F90
subroutine gqarcx	metlib4.F90
subroutine gqari	metlib4.F90
subroutine gqarid	metlib4.F90
subroutine gqaridx	metlib4.F90
subroutine gqarix	metlib4.F90
subroutine gqarr	metlib4.F90
subroutine gqarrd	metlib4.F90
subroutine gqarrdx	metlib4.F90
subroutine gqarrx	metlib4.F90
subroutine gqexv	metlib4.F90
subroutine helplevel	metlib4.F90
subroutine hpcalc	metlib4.F90
subroutine hphelp	metlib4.F90
subroutine iniio	metlib4.F90
subroutine init_help	metlib4.F90
subroutine loadc	metlib4.F90
subroutine loadr	metlib4.F90

Name	File
subroutine loadr1	metlib4.F90
subroutine loadrn	metlib4.F90
subroutine macbeg	metlib4.F90
subroutine macend	metlib4.F90
subroutine nghist	metlib4.F90
subroutine nohelp	metlib4.F90
subroutine nybin	metlib4.F90
subroutine nydat	metlib4.F90
subroutine nylp	metlib4.F90
subroutine nyrp	metlib4.F90
subroutine nyuni	metlib4.F90
subroutine nyvar	metlib4.F90
subroutine openlogfile	metlib4.F90
subroutine putfun	metlib4.F90
subroutine putprp	metlib4.F90
subroutine q1help	metlib4.F90
subroutine q2help	metlib4.F90
subroutine q3help	metlib4.F90
subroutine q3helpx	metlib4.F90
subroutine q4help	metlib4.F90
subroutine set_echo	metlib4.F90
subroutine sortin	metlib4.F90
subroutine sortrd	metlib4.F90
subroutine sortrdd	metlib4.F90
subroutine ssort	metlib4.F90
subroutine storc	metlib4.F90
subroutine storr	metlib4.F90
subroutine storr1	metlib4.F90
subroutine storrn	metlib4.F90
subroutine tophlp	metlib4.F90
subroutine untab	metlib4.F90
subroutine winit	metlib4.F90
subroutine wlist	metlib4.F90
subroutine wold	metlib4.F90
subroutine wpatch	metlib4.F90
subroutine wphlp	metlib4.F90
subroutine wrels	metlib4.F90
subroutine wrice	metlib4.F90
subroutine wrice2	metlib4.F90
subroutine wrihex	metlib4.F90
subroutine wriint	metlib4.F90

Name	File
subroutine wrinum	metlib4.F90
subroutine wrkchk	metlib4.F90
subroutine wrtbiq	metlib4.F90
subroutine wrtdaq	metlib4.F90
subroutine wrtfun	metlib4.F90
subroutine wrtlpq	metlib4.F90
subroutine wrtrpq	metlib4.F90
subroutine wsave	metlib4.F90
subroutine wtake	metlib4.F90
subroutine wtrest	metlib4.F90

References

[1] B. Sundman, PhD thesis, KTH, Stockolm, Sweden