

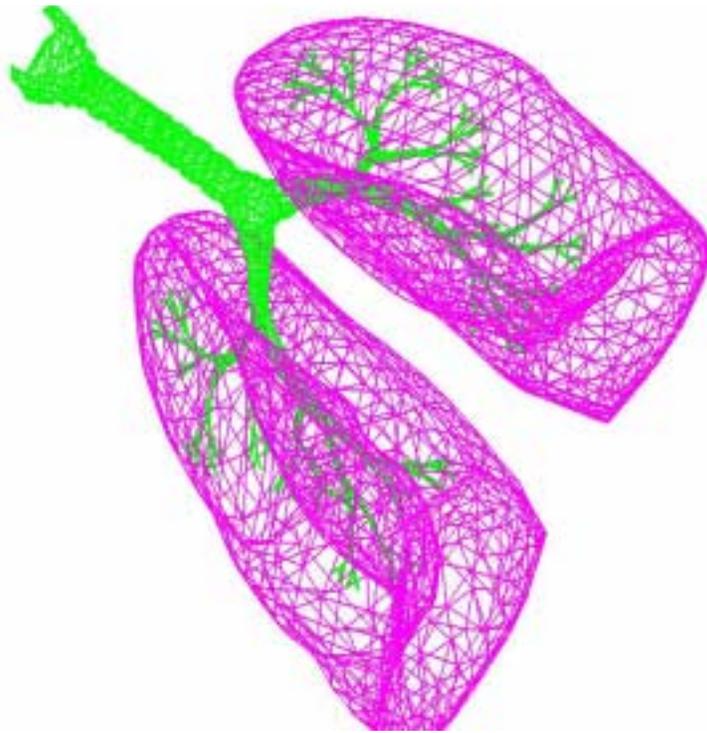
# *NWGrid*

# Programmers Reference

## The NorthWest *Grid* Generation Code

---

Web Site: <http://www.emsl.pnl.gov/nwgrid>



---

**Dr. Harold E. Trease**  
**Lynn L. Trease**  
**Applied Mathematics Group**  
**Theory, Modeling, and Simulation Directorate**  
*William R. Wiley, Environmental Molecular Sciences Laboratory*  
**Pacific Northwest National Laboratory**  
*Operated by Battelle for the US Department of Energy*

**TABLE OF CONTENTS**

1.0 Building an Executable .....3

2.0 Issuing Commands from a User Program.....4

3.0 Writing User Commands .....5

4.0 Accessing the Mesh Object.....6

5.0 Utility Subroutines .....8

    5.1 Memory Manager .....8

    5.2 Mesh Objects .....10

    5.3 Point Selection .....14

    5.4 Character Length.....16

    5.5 Retrieving Point Sets and Element Sets.....17

    5.6 Array Compression .....18

    5.7 Array Sorting .....18

    5.8 Miscellaneous .....20

    5.9 Geometry Information .....21

6.0 Documentation.....23

7.0 Contact .....24

## 1.0 Building an Executable

The executable is built by linking a driver routine with the code and utility libraries. The driver routine must contain a call to `initx3d` and a call to `dotask` and must contain a subroutine called `user_sub`. A sample driver routine is listed:

```

        program adrivgen
C
C #####
C
C PURPOSE -NWGrid driver
C #####
C
C      implicit real*8 (a-h,o-z)
C
C      call initNWGrid('generate','noisy',';')
C
C      call control_command_lg(ierr_return)
C
C      stop
C      end
C
C      subroutine user_sub(imsgin,xmsgin,cmsgin,msgtyp,nwds,ierr1)
C #####
C
C PURPOSE -
C Process user supplied commands
C
C INPUT ARGUMENTS -
C
C imsgin - integer array of tokens returned by parser
C xmsgin - real array of tokens returned by parser
C cmsgin - character array of tokens returned by parser
C msgtyp - int array of token types returned by parser
C nwds - number of tokens returned by parser
C
C OUTPUT ARGUMENTS -
C
C ierr1 - 0 for successful completion - -1 otherwise
C #####
C      character*32 cmsgin(nwds)
C      integer imsgin(nwds),msgtyp(nwds)
C      integer nwds,ierr1,lenc
C      real*8 xmsgin(nwds)
C get command length
C      lenc=icharlnf(cmsgin(1))
C set default error return to fail
C      ierr1=-1
C Insert code here to handle user coded subroutines
C For example
C      if(cmsgin(1)(1:lenc).eq.'my_cmnd') then
C          call my_rtn(imsgin,xmsgin
C *          cmsgin,msgtyp,nwds,ierr1)
C      else
C          ierr1=-1
C      endif
C
C      return
C      end

```

Sample build scripts for the supported (IBM and SGI) and unsupported platforms are:

### IBM

```
xlf -g -o nwgridgen -qintlog -qcharlen=500 -bname:.fdate,.fdate_ adrivgen.f lib-  
nwgrid.a libutil.a
```

### SGI

```
f90 -O2 -n32 -r10000 -o nwgridgen adrivgen.f libnwgrid.a libutil.a
```

### Sun OS and Sun Solaris

```
f90 -O2 -o nwgridgen adrivgen.f libnwgrid.a libutil.a
```

### HP

```
f90 +U77 -R8 -lm -o nwgridgen adrivgen.f libnwgrid.a libutil.a
```

### DEC

```
f90 -fast -arch ev56 -i8 -o nwgridgen adrivgen.f libnwgrid.a libutil.a
```

Note: The NWGrid Dictionary must also be installed. See the NWGrid Installation Guide.

## 2.0 Issuing Commands from a User Program.

Any NWGrid command can be issued by calling the subroutine `dotask`, for example:

```
call dotask ('cmo/select/3dmesh; finish', ier1)
```

will select the Mesh Object named *3dmesh*. `ier1` will be zero if the commands are executed with no error, non-zero otherwise. A sequence of commands can also be executed using `dotask`. For example:

```
call dotask ('cmo/addatt/cmo; cmo/list;finish',ier1)
```

will execute the two commands and then return to the next executable statement. A 'finish' command must always be the last command in the command list for `dotask`. The user should also be aware that mesh object points might be changed by the `dotask` commands and should be refreshed using `cmo_get_info`. The parser arrays `cmsgin`, `xmsgin` etc. will not be the same upon return from *dotask*.

By using the NWGrid command **infile**, a series of commands may be executed, for example

```
call dotask ('infile/mydeck; finish', ier1)
```

will execute all the NWGrid commands that are in the user's file named *mydeck*. The final command in the file *mydeck* should be **finish**. *mydeck* itself may contain additional **infile** commands.

### 3.0 Writing User Commands

The access to user written subroutines is through the NWGrid subroutine, `user_sub`. It is passed the parsed command input line. The parser breaks up the input line into tokens and returns to NWGrid a count of number of tokens, an array containing the token types, and the tokens themselves. The parameters returned by the parser are:

Parameters	I/O	Description/Default
nwds	O	number of tokens
msgtyp	O	integer array of token types - 1 for integer, 2 for real, 3 for character, msgtyp(nwds+1) = -1
imsgin	O	array of integer tokens, e.g. imsgin(i) is the ith token which is an integer if msgtyp(i)=1
xmsgin	O	array of real tokens
cmsgin	O	array of character tokens

Null fields are given the integer value 0, real value 0. and character value '-def-'. The parser is written in C, therefore character variables returned will be null terminated on some platforms. A FORTRAN function is supplied; `icharlnf` will return the length of the character string blank or null terminated, ignoring leading blanks

If the user has written a subroutine, `my_routine`, that responds to the command, `my_comnd`, the call from `user_sub` should look like:

```

c      elseif (cmsgin(1)(1:lenc).eq. 'my_comnd')
c          call my_routine(nwds,imsgin,xmsgin,cmsgin,msgtyp,ierr1)

```

The subroutine `my_routine` should set `ierr1` to zero if the command is processed successfully and should use the `cmo` interface routines to access the components of the Mesh Object that it needs, for example:

```

character*32 cmo
pointer (ipimt1, imt1(*))

```

- c get the name of the current mesh object  
call `cmo_get_name(cmo_name,ierror)`
- c get the number of nodes and the material ids  
call `cmo_get_info('nnodes',cmo_name,nnodes,ilen,ityp,ierr)`  
call `cmo_get_info('imt1',cmo_name,ipimt1,ilen,ityp,ierr)`

The subroutine `user_sub` is supplied with the driver and is defaulted to print the error message: 'Illegal command' and return.

### 4.0 Accessing the Mesh Object

The following template is an example of using the an existing mesh object and of creating a new mesh object. The existing mesh object is a 3d object. The object to be created is a 2d object. It is first necessary to set up the pointer statements for both the existing and new mesh objects. All mesh object attributes are integers except for `xic,yix,zic` which are `real*8`.

```
C Definitions for incoming (existing) cmo
C
  pointer (ipimt1, imt1)
  pointer (ipitp1, itp1)
  pointer (ipicr1, icr1)
  pointer (ipisn1, isn1)
  integer imt1(1000000), itp1(1000000),
*   icr1(1000000), isn1(1000000)
  pointer (ipxic, xic)
  pointer (ipyic, yic)
  pointer (ipzic, zic)
  dimension xic(1000000), yic(1000000), zic(1000000)
  pointer (ipitetclr, itetclr)
  pointer (ipitettyp, itettyp)
  pointer (ipitetoff, itetoff)
  pointer (ipjtetoff, jtetoff)
  pointer (ipitet, itet)
  pointer (ipjtet, jtet)
  integer itetclr(1000000), itettyp(1000000),
*   itetoff(1000000), jtetoff(1000000)
  integer itet(4,1000000) , jtet(4,1000000)
C
C Definitions for cmo that is to be created
C
  pointer (ipimt1a, imt1a) pointer (ipitp1a, itp1a)
  pointer (ipicr1a, icr1a)
  pointer (ipisn1a, isn1a)
  integer imt1a(1000000), itp1a(1000000),
*   icr1a(1000000), isn1a(1000000)
  pointer (ipxica, xica)
  pointer (ipyica, yica)
  pointer (ipzica, zica)
  dimension xica(1000000), yica(1000000), zica(1000000)
  pointer (ipitetclra, itetclra)
  pointer (ipitettypa, itettypa)
  pointer (ipitetoffa, itetoffa)
  pointer (ipjtetoffa, jtetoffa)
  pointer (ipiteta, iteta)
  pointer (ipjteta, jteta)
  integer itetclra(1000000), itettypa(1000000),
*   itetoffa(1000000), jtetoffa(1000000)
  integer iteta(3,1000000) , jteta(3,1000000)
C Get the existing cmo - its name is in the variable cmoin
```

```

C
    call cmo_get_name(cmoin,ier)
C
C Get the scalar mesh variables
    call cmo_get_info('nnodes',cmoin,npoints,lencm,itypcm,ier)
    call cmo_get_info('nelements',cmoin,ntets,lencm,itypcm,ier)
    call cmo_get_info('ndimensions_topo',cmoin,ndt,lencm,itypcm,ier)
    call cmo_get_info('ndimensions_geom',cmoin,ndg,lencm,itypcm,ier)
    call cmo_get_info('nodes_per_element',cmoin,npe,lencm,itypcm,ier)
    call cmo_get_info('faces_per_element',cmoin,nfpe,lencm,itypcm,ier)
    call cmo_get_info('mbndry',cmoin,mbndry,lencm,itypcm,ier)
C
C Get pointers to the vector variables
    call cmo_get_info('ialias',cmoin,ipialias,lenialias,ictype,ier)
    call cmo_get_info('imt1',cmoin,ipimt1,lenimt1,ictype,ier)
    call cmo_get_info('itp1',cmoin,ipitp1,lenitp1,ictype,ier)
    call cmo_get_info('icr1',cmoin,ipicr1,lenicr1,ictype,ier)
    call cmo_get_info('isn1',cmoin,ipisn1,lenisn1,ictype,ier)
    call cmo_get_info('xic',cmoin,ipxic,lenxic,ictype,ier)
    call cmo_get_info('yic',cmoin,ipyic,lenyic,ictype,ier)
    call cmo_get_info('zic',cmoin,ipzic,lenzic,ictype,ier)
    call cmo_get_info('itetclr',cmoin,ipitetclr,lenitetclr,ictype,ier)
    call cmo_get_info('itettyp',cmoin,ipitettyp,lenitettyp,ictype,ier)
    call cmo_get_info('itetoff',cmoin,ipitetoff,lenitetoff,ictype,ier)
    call cmo_get_info('jtetoff',cmoin,ipjtetoff,lenjtetoff,ictype,ier)
    call cmo_get_info('itet',cmoin,ipitet,lenitet,ictype,ier)
    call cmo_get_info('jtet',cmoin,ipjtet,lenjtet,icmotype,ier)
C
C Create the new 2d cmo - call it cmoout.
C
    call cmo_exist(cmoout,ier)
C
C ier.eq.0 means that the cmo already exists - if so release it.
C
    if(ier.eq.0) call cmo_release(cmoout,idelete)
C
C Set active cmo to cmoout
    call cmo_set_name(cmoout,ier)
C
C set scalar mesh variables
C
    call cmo_set_info('nnodes',cmoout,npoints,1,1,ier)
    call cmo_set_info('nelements',cmoout,ntets,1,1,ier)
C
C the following scalars need to be set for a 2d cmo
C
    call cmo_set_info('ndimensions_topo',cmoout,2,1,1,ier)
    call cmo_set_info('ndimensions_geom',cmoout,3,1,1,ier)
    call cmo_set_info('nodes_per_element',cmoout,3,1,1,ier)
    call cmo_set_info('faces_per_element',cmoout,3,1,1,ier)
C
C allocate memory for vector variables
    call cmo_newlen(cmoout,ier)
C
C now get the pointers to the allocated memory for the vector data
    call cmo_get_info('imt1',cmoout,ipimt1a,lenimt1a,icmotype,ier)
    call cmo_get_info('itp1',cmoout,ipitp1a,lenitp1a,icmotype,ier)
    call cmo_get_info('icr1',cmoout,ipicr1a,lenicr1a,icmotype,ier)
    call cmo_get_info('isn1',cmoout,ipisn1a,lenisn1a,icmotype,ier)
    call cmo_get_info('xic',cmoout,ipxica,lenxica,icmotype,ier)
    call cmo_get_info('yic',cmoout,ipyica,lenyica,icmotype,ier)
    call cmo_get_info('zic',cmoout,ipzica,lenzica,icmotype,ier)
    call cmo_get_info('itetclr',cmoout,ipitetclra,lenclra,icmotype,ier)
    call cmo_get_info('itettyp',cmoout,ipitettypa,lentypa,icmotype,ier)

```

```
call cmo_get_info('itetoff',cmoout,ipitetoffa,lenoffa,icmotype,ier)
call cmo_get_info('jtetoff',cmoout,ipjtetoffa,lenoffa,icmotype,ier)
call cmo_get_info('itet',cmoout,ipiteta,leniteta,icmotype,ier)
call cmo_get_info('jtet',cmoout,ipjteta,lenjteta,icmotype,ier)
```

C

C now the values for the vector components of the 2d mesh  
C object can be set.

### 5.0 Utility Subroutines

The following subroutines are available to code developers who wish to add modules to NWGrid.

- Memory Manager (mmgetblk, mmrelblk, mmrelprt, mmincblk, mmfindbk, mmgettyp, mmgetlen, mmgetnam, mmprint, mmverify, mmgetbk)
- Mesh Objects (cmo\_create, cmo\_get\_info, cmo\_set\_info, cmo\_get\_name, cmo\_set\_name, cmo\_get\_attribute\_name, cmo\_newlen, cmo\_release, get\_info\_c, get\_info\_i, get\_info\_r)
- Point Selection (getptyp, unpackpc, unpacktp)
- Character Length (icharln, icharlnf, icharlnb)
- Retrieving Point Sets and Element Sets (eltlimc, pntlimc, pntlimn)
- Array Compression (kmprsm, kmprsn, kmprsnr, kmprsnrrr, kmprsp, kmprspr, kmprsz, kmprszr)
- Array Sorting (hpsort, hpsorti, hpsortim, hpsortimp)
- Miscellaneous (setsize, set\_user\_bounds, inside)
- Geometry Information (get\_regions, get\_mregions, get\_surfaces, hgetprt)

### 5.1 Memory Manager

NWGrid uses dynamic memory allocation. Memory is referenced by a two part name, block name and partition name. It is allocated in integer or real blocks (real is implemented as real\*8). Each memory block is preceded by a header and terminated by a trailer. The memory manager always returns the pointer to the data section of the memory block. Length is specified in words. Type indicates if the words are integer or real. Different platforms will have different values for integer and real word lengths. These machine dependent values are collected in the include file machine.h.

**mmgetblk** (blkln, prtln, iadr, length, itype, icrcode)

Allocate a block of memory.

Parameters	I/O	Description/Default
blkln	I	Block name of memory block.
prtln	I	Partition name of memory block.
iadr	O	pointer to memory block (data section)
length	I	number of words to be allocated
itype	I	1 for integer, 2 for real (real*8)
icscode	O	return code, 0 for no errors

**mmrelblk** (blkln, prtln, iadr, icscode)

Release a block of memory. (Note: iadr is not used.)

**mmrelprt** (prtln, icscode)

Release a partition of memory -- all blocks belonging to this partition will be released.

**mmincblk** (blkln, prtln, iadr, increment, icscode)

Increment a block of memory.

Parameters	I/O	Description/Default
increment	I	number of words to increment memory block

**mmfindbk** (blkln, prtln, iadr, length, icscode)

Find pointer to a block of memory.

**mmgettyp** (ipin, itypout, icscode)

Find pointer to a block of memory.

Parameters	I/O	Description/Default
ipin	I	pointer to memory block (data)
itypout	O	type of data 1 for integer, 2 for real

**mmgetlen** (ipin, lenout, icrcode)

Return number of words in a block of memory.

<b>Parameters</b>	<b>I/O</b>	<b>Description/Default</b>
lenout	O	number of words in a memory block

**mmgetnam** (ipin, blkout, prtout, icrcode)

Return name of a block of memory.

<b>Parameters</b>	<b>I/O</b>	<b>Description/Default</b>
blkout	O	block name
prtout	O	partition name

**mmprint** ()

Print a dump of allocated memory. This is useful for debugging purposes; the dump is listed in two parts, by time of allocation and by increasing pointer address.

**mmverify** ()

Verify memory integrity. Print debug information if the memory block headers or trailers have been overwritten.

**mmgetbk** (blkln, prtln, iadr, length, itype, icrcode)

Find a block of memory, increase it if shorter than length, or allocate it if it doesn't exist.

## 5.2 Mesh Objects

**cmo\_create** (cmo\_name, ierror)

Create a new mesh object called cmo\_name.

<b>Parameters</b>	<b>I/O</b>	<b>Description/Default</b>
cmo_name	I	name of mesh object
ierror	O	error return - 0 if no errors

**cmo\_get\_info** (ioption, cmo\_name, iout, lout, itype, ierror)

Get values of scalar attribute of the mesh object. Get pointers to vector attributes.

Parameters	I/O	Type/Value	Description/Default
ioption	I		Name of mesh object attribute whose value is to be retrieved; the information retrieved may be one of these key words or it may be the name of a user supplied attribute (generated by a <b>cmo/addatt</b> command):
		<b>number_of_attributes</b>	
		<b>nnodes</b>	number of nodes in the mesh
		<b>nelements</b>	number of elements in the mesh
		<b>nfaces</b>	number of unique topological facets
		<b>nedges</b>	number of unique edges in mesh
		<b>mbndry</b>	boundary node flag value
		<b>ndimensions_topo</b>	topological dimensionality
		<b>ndimensions_geom</b>	
		<b>nodes_per_element</b>	
		<b>edges_per_element</b>	
		<b>faces_per_element</b>	
		<b>isetwd</b>	pset membership information
		<b>ialias</b>	alternate node numbers
		<b>imt1</b>	node material
		<b>itp1</b>	node type
		<b>icr1</b>	constraint numbers for nodes
		<b>isn1</b>	child, parent node correspondence

<b>Parameters</b>	<b>I/O</b>	<b>Type/Value</b>	<b>Description/Default</b>
		<b>ign1</b>	igeneration numbers for nodes
		<b>xic, yic, zic</b>	node coordinates
		<b>itetclr</b>	integer array of element material
		<b>itettyp</b>	geometry of element
		<b>xtetwd</b>	eltset membership information
		<b>itetoff</b>	index into itet array for an element
		<b>jtetoff</b>	index into jtet array for an element
		<b>itet</b>	node vertices for each element
		<b>jtet</b>	element connectivity
cmo_name	I		name of mesh object to be retrieved
iout	O		value of attribute if the attribute is a scalar or a pointer to the attribute if the attribute is a vector
lout	O		2 for mesh object array , 1 for mesh object scalar
itype	O		9999 for mesh object array,1 for mesh object scalar, -1 for error
ierror	O		return flag ( 0 if no errors)

**cmo\_set\_info** (ioption, cmo\_name, data, lin, itype, ierror)

Set values of scalar attribute of the mesh object. Vector attributes are set by filling arrays pointed to by the vector attribute pointer.

<b>Parameters</b>	<b>I/O</b>	<b>Description/Default</b>
ioption	I	attribute name
data	I	value of the scalar attribute to be set
lin	I	length code of attribute (1)

Parameters	I/O	Description/Default
itype	I	type of attribute (1)

**cmo\_get\_name** (cmo\_name, ierror)

Get the name of the current mesh object. cmo-name name of current mesh object

**cmo\_set\_name** (cmo\_name, ierror)

Set the name of the current mesh object.

**cmo\_get\_attribute\_name** (cmo\_name, attribute-index, attribute-name, ierror)

This routine is useful when looping through all the attributes of a mesh object. To get the number of attributes use `cmo_get_info(number_of_attributes,...`

Parameters	I/O	Description/Default
attribute-index	I	number of attribute
attribute-name	I	name of retrieved attribute

**cmo\_newlen** (cmo\_name, ierror)

Adjust memory associated with mesh object. Must be called whenever the size of the mesh is adjusted in order to provide memory to the pointered attributes.

**cmo\_release** (cmo\_name, ierror)

Release a mesh object called cmo-name and release its memory.

**get\_info\_c** (attribute or parameter-name, cmo\_name, storage\_block\_name, class\_name, cdata, ierror)

Retrieve a mesh object parameter value that is retrieve character data.

<b>Parameters</b>	<b>I/O</b>	<b>Description/Default</b>
attribute or parameter name	I	name of mesh object attribute or global parameter
cmo_name	I	name of mesh object, ignored for global parameters.
storage-block-name	I	sbglobal for global parameters, & sbcmoatt for mesh object attributes
class_name	I	default for global and mesh object field names - & field name such as rank, type' for mesh object attributes
cdata,idata,rdata	O	returned value of parameter
ierror	O	error return - 0 if no errors

**get\_info\_i** (attribute or parameter-name, cmo\_name, storage\_block\_name, class\_name, idata, ierror)

Retrieve a mesh object parameter value that is integer data.

**get\_info\_r** (attribute or parameter-name,cmo\_name, storage\_block\_name, class\_name, rdata, ierror)

Retrieve a mesh object parameter value that is real data.

### Examples

call **get\_info\_i**('ipointi',cmo,'sbglobal','default', ipointi, ierror)  
 retrieve the value of the global parameter ipointi and store it in the variable ipointi

call **get\_info\_c**('xic', cmo, 'sbcmoatt','rank',crank,ierror)  
 Will retrieve the rank of the mesh object attribute xic and place it in crank. The default value of the rank of xic is SCALAR but might have been modified.

call **get\_info\_i**(crank,cmo,'sbcmoatt','default',irank,ierror)  
 Will retrieve the integer value of the rank of the attribute

accessed in the previous call(i.e. if crank = SCALAR, then irank=1, unless SCALAR has been redefined by the user).

call **cmo\_get\_length**(ipcmosb,att\_name,length,irank,ierror)  
Retrieves the integer values of the requested mesh object attributes length and rank.

### 5.3 Point Selection

**getptyp** (point\_type\_name, point\_type, ierror)

This routine converts point type names to point types. See the NWGrid Data Structures Document for a list of point types, names and meanings.

Parameters	I/O	Description/Default
point_type_name	I	name of point type
point_type	O	value of point type
ierror	O	error return

**unpackpc** (npoints, itp1, isn1, iparents)

This routine returns in the array iparents the parent point corresponding to each child point i, if point i is a child point. Ordinary points are their own parents.

Parameters	I/O	Description/Default
npoints	I	number of nodes
itp1	I	array of point types
isn1	I	array of parent child links
iparents	O	array of parent node number for each point

**unpacktp** (ioptip, iopt2, inum, ipitp1, ipitp2, ierror)

This routine sets, or's in, or and's in (depending on iopt2) a 1 in the array pointed to by ipit2 for each point that fits the criterion specified by ioptip. A zero is set, or'd or and'd otherwise.

Parameters	I/O	Type/Value	Description/Default
ioptip	I		criterion
		<b>allreal</b>	( $0 \leq \text{itp1}(i) \leq 19$ )
		<b>interior</b>	( $\text{itp1}(i)=0$ )
		<b>inteintf</b>	( $\text{itp1}(i)=2,3,4$ )
		<b>matlintr</b>	( $\text{itp1}(i)=2,4,8,9,12,13,15,19$ )
		<b>boundary</b>	( $8 \leq \text{itp1}(i) \leq 19$ )
		<b>reflect</b>	( $\text{itp1}(i)=9,10, 12, 14, 15,16,18,19$ )
		<b>free</b>	( $\text{itp1}(i)=8,9,11, 13, 14, 15,17,18$ )
		<b>intrface</b>	( $\text{itp1}(i)=2,3,4,8,9,12, 13,15,16,17,18,19$ )
		<b>virtual</b>	( $\text{itp}(i)=3,4,8,9,16,17,18,19$ )
		<b>removed</b>	( $20 \leq \text{itp1}(i) \leq 29$ )
		<b>merged</b>	( $\text{itp1}(i)=20$ )
		<b>dudded</b>	( $\text{itp1}(i)=21$ )
opt2	I		operation
		<b>set</b>	set itp2 to 1 or 0
		<b>or</b>	or in a 1 or 0 in itp2
		<b>and</b>	and in a 1 or 0 in itp2
inum	I		number of nodes in itp1 array
ipitp1	I		pointer to array of point types
ipitp2	O		pointer to output array of 1's or 0's (length inum)
ierror	O		error message

## 5.4 Character Length

NWGrid uses a parser written in C whereas most other modules are written in FORTRAN, the user must be very careful in using character comparison. Some character strings will be terminated with a blank (FORTRAN) and some by a null (C). The following functions are provided to return character string length (number of characters in `iword` ignoring terminator character).

### **icharln** (`iword`)

Search for terminating blank or null.

### **icharlnf** (`iword`)

Ignore leading blanks then search for terminating blank or null.

### **icharlnb** (`iword`)

Search backwards for first blank or null - uses FORTRAN function **len** to give starting point (this is a risky assumption)

## 5.5 Retrieving Point Sets and Element Sets

### **eltlimc** (`ich1, ich2, ich3, ipmary, mpno, ntets, xtetwd`)

`eltlimc` returns an array of element numbers where the elements belong to the `eset` given in the argument list. `Esets` must be specified by name. On return the array pointed to by `ipmary` will contain the `mpno` element numbers that belong to the `eset`.

Parameters	I/O	Description/Default
<code>ich1,ich2,ich3</code>	I	<code>eset,get,eset_name</code>
<code>ipmary</code>	I	pointer to array of elements of <code>eset_name</code>
<code>mpno</code>	I	number of elements in <code>eset_name</code>
<code>ntets</code>	I	num of elements in mesh object
<code>xtetwd</code>	I	array of <code>eset</code> membership information

### **pntlimc** (`ich1, ich2, ich3, ipmary, mpno, npoints, isetwd, itp1`)

**pntlmc**,**pntlimn** return an array of node numbers where the nodes belong to the **pset** given in the argument list. On return the array pointed to by **ipmpary** will contain **mpno** node numbers. These numbers are the nodes that belong to the **pset**.

Parameters	I/O	Description/Default
ich1,ich2,ich3	I	<b>pset,get,pset_name</b>
ipmpary	I	pointer to array of node number of pset_name
mpno	I	number of nodes in pset_name
npoints	I	numer of nodes in mesh object
isetwd	I	array of pset membership information
itp1	I	array of point types

**pntlmc** (ich1, ich2, ich3, ipmary, mpno, npoints, isetwd, itp1)

## 5.6 Array Compression

The following utility routines compress arrays. Note that the output array may be the same as the input array in which case the compression is done in place. Also the mask array may be the same as the input array. The name suffixes of the compression routine may be decoded as **m** minus (negative), **n** non-zero, **p** positive, **z** equal to zero. If the routine name ends in **rrr**, the mask, input and output arrays are all real. If the name ends in a single **r**, the mask is real, the input and output arrays are integers. Otherwise the mask, input and output arrays are all integers. For example **kmprsn**(100,int,1,int,1,int,1,num) will compress all the zeros out of array **int**.

- kmprsm** (n, z, iz, x, ix, , y, iy, count)
- kmprsn** (n, z, iz, x, ix, , y, iy, count)
- kmprsnr** (n, z, iz, x, ix, , y, iy, count)
- kmprsnrrr** (n, z, iz, x, ix, , y, iy, count)
- kmprsp** (n, z, iz, x, ix, , y, iy, count)
- kmprspr** (n, z, iz, x, ix, , y, iy, count)
- kmprsz** (n, z, iz, x, ix, , y, iy, count)

**kmprszr** (n, z, iz, x, ix, , y, iy, count)

Parameters	I/O	Description/Default
n	I	length of z and x
z	I	array of masks
iz	I	stride in z
x	I	array of source
ix	I	stride of x
y	I	array of output
iy	I	stride in y
count	O	length of y

## 5.7 Array sorting

The following utility routines sort arrays.

**hpsort** (n, ra, ascend, iprm)

Parameters	I/O	Description/Default
n	I	number of elements to be sorted
ra	I	a real array
ascend	I	real which controls direction of sort
iprm	I	integer array to be reordered

**hpsorti** (n, ia)

Parameters	I/O	Description/Default
n	I	number of elements to be sorted
ia	I	integer array to be sorted

**hpsortim** (n, m, md, itemp, ia)

Parameters	I/O	Description/Default
n	I	number of columns to sort into ascending order
m	I	maximum number of keys (rows) to consult for comparisons
md	I	column length of array IA ( $M \leq MD$ )
itemp	I	temp array of length MD.
ia	I	integer array of MD-tuples to be reordered

**hpsortimp** (n, m, md, ia, ascend, iprm)

Parameters	I/O	Description/Default
n	I	number of elements to be sorted
m	I	we interpret array IA as m-tuples
md	I	actual first dimension of array IA
ia	I	integer array of values which determines how IPRM will be reordered
ascend	I	real*8 which controls direction of sort
iprm	I	integer array to be reordered

## 5.8 Miscellaneous

**setsize** (xmin, xmax, ymin, ymax, zmin, zmax, epsilon, epsilonv)

The subroutine **setsize** set the code variables, xmin, xmax, ymin, ymax, zmin, zmax, epsilon, epsilonv

Parameters	I/O	Description/Default
xmin, xmax, ymin, ymax, zmin, zmax	I	set from the minimum and maximum xic,yic,zic values of all 'real' points (duded and merged points will be ignored).
epsilon	O	$((xmax-xmin)**2+(ymax-ymin)**2+(zmax-zmin)**2) * epsilon_r * 10$

Parameters	I/O	Description/Default
epsilon	O	$\text{abs}(\text{xmax}-\text{xmin}) * \text{abs}(\text{ymax}-\text{ymin}) * \text{abs}(\text{zmax}-\text{zmin}) * \text{epsilon} * 10$

**getsize** (xmin, xmax, ymin, ymax, zmin, zmax, epsilon, epsilon)

Return the values of the code variables set by the command **setsize**

**set\_user\_bounds** (nubndpts, ubndpts, cmo, ipattr, idfield)

This routine allows the user to set boundary values.

Parameters	I/O	Description/Default
<b>nubndpts</b>	I	number of boundary nodes
<b>ubndpt</b>	I	integer array of boundary node indices
<b>cmo</b>	I	a mesh object name
<b>ipattr</b>	I	pointer to mesh object attribute to contain boundary values
<b>idfield</b>	I	identifier used to identify the set of boundary nodes.

The **inside** set of subroutines test whether a query point is strictly inside, strictly outside or on the surface of the specified element. The value returned in iflag is 0 if the query point is inside the element, -1 if outside, or is set to the local face number containing the query point. Coordinates of the query point are in xq, yq, zq. Coordinates of the vertices of the element are in x1, y1, z1, x2,.... The coordinates of the element must be specified in the correct order

**inside\_pyr**(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,xq,yq,zq,iflag)

**inside\_pri**(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,x6,y6,z6,xq,yq,zq,iflag)

**inside\_hex**(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,x6,y6,z6,x7,y7,z7,x8,y8,z8,xq,yq,zq,iflag)

**inside\_tet**(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,xq,yq,zq,iflag)

ag)

**inside\_quad2d**(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,xq,yq,zq,iflag)

**inside\_tri2d**(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,xq,yq,zq,iflag)

These routines should not be confused with

**inside\_quad**(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,xq,yq,zq,iflag)

**inside\_tri**(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,xq,yq,zq,iflag)

which return in iflag: 1 if the query point is on the plane of the element, 0 if below the plane and -1 if above.

## 5.9 Geometry Information

Much geometry information is maintained in the **geom** common block which is defined in the include file **geom.h**.

Parameters	I/O	Description/Default
nsurf		is a pointer to the array of pointers to the region definitions. See subroutine <b>dumpgeom</b> for an example of decoding this information.
nregs		is a pointer to an array of region names
nmregs		is a pointer to an array of lengths of region definitions (i.e. number of tokens in the definition)
maxdef		is the number of tokens in the longest region definition
maxindef		is the number of tokens in the longest mregion definition.

**get\_regions** (ipregs, ipcregs, ipndefregs)

This routine will update nregs and maxdef in the **geom** common block.

Parameters	I/O	Description/Default
ipregs	O	is a pointer to the array of pointers to the region definitions. See subroutine <b>dumpgeom</b> for an example of decoding this information.
ipcregs	O	is a pointer to an array of region names
ipndefregs	O	is a pointer to an array of lengths of region definitions (i.e. number of tokens in the definition)

**get\_mregions** (ipregs, ipmregs, ipnndefregs, ipmatregs)

This routine is analogous to get\_regions but has an additional argument.

Parameters	I/O	Description/Default
ipmatregs	O	pointer to an array of material numbers for each material region.

**get\_surfaces** (ipisall, ipisatt, ipcsall, ipistype, ipibtype)

This routine will update nsurf.

Parameters	I/O	Description/Default
ipisall	O	is a pointer to first part of surface data (coordinate information)
ipisatt	O	is a pointer to the second part of the surface date (type and boundary type). See subroutine <b>dumpgeom</b> for an example of decoding this information
ipcsall	O	is a pointer to the array of surface names
ipistype	O	is a pointer to the array of surface types
ipibtype	O	is a pointer to the array of surface boundary types

**hgetprt** (iopt, ipsb, cpart, ipartname, imd, ierror)

This routine will retrieve the material number that corresponds to a material region and vice versa. Either *ipartname* or *imd* must be input; the other will be output depending on the value of *iopt*.

<b>Parameters</b>	<b>I/O</b>	<b>Description/Default</b>
<i>iopt</i>	I	1 to return material number 2 to return material region name
<i>ipsb</i>	I	pointer to `sbcmaprm' storage block
<i>cpart</i>	I	`part'
<i>ipartname</i>	O	material region name
<i>imd</i>	O	material number
<i>ierror</i>	O	error return flag

### **6.0 Documentation**

The following *NWGrid* documents can be found on the internal, *NWGrid/NWPhys* web site. <http://www.emsl.gov/nwgrid>

- NWGrid* Installation Guide
- NWGrid* Users Manual
- NWGrid* Tutorial
- NWGrid* Examples
- NWGrid* Command Reference
- NWGrid* Data Structures Reference

### **7.0 Contact**

Contact Lynn Trease, [llt@pnl.gov](mailto:llt@pnl.gov), for further assistance and questions.