setpar, getpar, mstpar, endpar − retrieve command-line arguments

The declaration for the argument depends on The routines provide a simple method to parse program arguments from the command line, and from files. Their use illustrated in the following example: main(ac,av)

```
int ac; char **av;
  {
        /* specify parameters, some with default values */
        static int nx =10;
        static char title[40] = "No title given";
        static float dx = 0.01;
        static char input[40];
        static float x[8];
        static int boo = 1; /* boolean, true */

        setpar(ac,av);                    /* initialize getpar */
        getpar("nx","d",&nx);
        getpar("dx","f",&dx);
        getpar("title","s",title);
        mstpar("input","s",input); /* must have this parameter */
        getpar("boo","b",&boo);
        getpar("x","vf[8]",x);
        endpar();                /* deactivate getpar */

        /* rest of program */
  }
```

The routine initializes the package. Its arguments are the same as those to main itself. It is an error to not call before any other getpar calls. The routines are deactivated with a call to This routine releases memory, and allows the option mentioned below to happen. The individual parameters are obtained with calls to and (must par) is identical to except that it terminates the program with an error message if the particular parameter is not specified. The following description of also applies to has three parameters. The first is a character string which specifies the external name of the parameter. It can be (practically speaking) of any length. The second parameter is a character string which specifies the type of variable the parameter is. Currently the following types are understood:

"d"	integer
"f"	float
"F"	double
"s"	character string
"b"	boolean (integer)
"vd"	integer vector
"vf"	float vector
"vF"	double vector

The *type* parameter can be used to indicate the maximum number of elements allowable in vectors. For example *type="vf[4]"* or *type="vf(4)"* would cause *getpar* tp modify no more than 4 elements of the vector, regardless of how many elements the user specifies. If no limit is specified, a limit of 10 is quietly enforced. The last parameter is a pointer to the type of variable indicated by does not modify this variable if no occurence of the parameter is found. Hence, a default value can be assigned before the call to returns 1 the parameter was found, and 0 if not. For vectors, returns the number of elements found. The parameters on the command line can occur in any order, and any number of times. In the case of multiple specifications, the last one is used. Any parameters that are not requested by or are ignored. An example of specifying parameters for the above program is: a.out dx=0.123 nx=300 title="sample title" dy=0.456 noboo x=1.0,4x2.0,2x5.12 Each specifcation is of the form No imbedded blanks are allowed on either side of the equals ("=") sign. Character strings with blanks or tabs are delimited with single (’) or double (") quotes. The only exceptions to the rule are boolean variables which are specifed as or to indicate true or false. Boolean variables may also be specified as integers with the form In the above example, true values for are specified as either or and false values as either or If is given then the returned value is The value for vector is given as a list separated by commas (,). No imbedded blanks are allowed in the list. Repetition factors (2x and 4x in the above example) can be used to specify repeated values. Several additional features are also available. At any point on the command line, the parameter can be given. This will cause to look in the file for additional parameters. Several arguments can be given on the command line. The search order is left to right. Consequently, any parameters given after the will override their values given in Also, the environment (if allowed, see NOENV option below) is searched first. Thus parameters on the command line and in par files override parameters set in the environment. The format of the parameters in the par file follow the same rules as the command line. Several specifications separated by while space, can occur on a given line, and their can be any number of lines. A ’#’ symbol in the position where a name would normally occur, indicates that the rest of the line is a comment, and is consequently ignored. The specification is also allow in the file, however recursions are limited in depth (current limit is 4). As a concession to the traditional switch passing method, a parameter of the form a.out -abc is available to the calling program as a character string with the call getpar("SWITCH","s",&sw); where in the example above, the string would be Parameters in the shell environment can be set (unset) with the C-shell commands: setenv name value unsetenv name The parameters in the environment can be printed with the command printenv Five additional parameters allow for input checking, and program interogation. The call to will terminate the program if this parameter is given. Each call to or will cause the name, type, and value of the variable to be listed on If is given, the listing is put in the file This option is useful for interogating a program as to what it wants for input. will list all input parameters that are found. This option is useful for debugging input data, and determining where a particular parameter is coming from in multiple specifications. If is given, the listing is put in the file will disallow any parameters to be obtained from the environment. It may occur on the command line, in a par file, or in the environment itself. will cause to print the name of the parameter before starting to search for it. This provides a quick method of determining which subroutine call is at fault, when a program dies in the getpar package. The routines are in the library and may be loaded with getlist(3), getarg(3) Be careful that is correctly specified, when dealing with floats and doubles. If a double pointer is used with *type="f"*, The lowest 32 bits of the mantissa will not be set correctly. If a float pointer and *type="F"* are used, the next element in memory will be clobbered. If the last combination is used in vector mode, you will get garbage back. If a program appears to be behaving differently for identical input parameters, make sure that some defaulted parameters are not sneaking in via the environment. The NOENV option may be of some use in this case. If you believe back door parameters are a bad practice, then set NOENV in your login shell. The routines produce error messages of the type:

```
****** ERROR program[getpar]: ******
        error message
```

Hopefully the error message is diagnostic of the trouble. If you are debugging a program try the VERBOSE option to see which call generated the problem. Errors in calls to are often reported as calls to Robert W. Clayton, Seismological Laboratory, Caltech, Pasadena, CA 91125 setpar, getpar, mstpar, endpar − retrieve command-line arguments

The declaration for the argument depends on The routines provide a simple method to parse program arguments from the command line, and from files. Their use illustrated in the following example: main(ac,av)

```
int ac; char **av;
  {
        /* specify parameters, some with default values */
        static int nx =10;
        static char title[40] = "No title given";
        static float dx = 0.01;
        static char input[40];
        static float x[8];
        static int boo = 1; /* boolean, true */

        setpar(ac,av);                          /* initialize getpar */
        getpar("nx","d",&nx);
        getpar("dx","f",&dx);
        getpar("title","s",title);
        mstpar("input","s",input); /* must have this parameter */
        getpar("boo","b",&boo);
        getpar("x","vf[8]",x);
        endpar();                     /* deactivate getpar */

        /* rest of program */
  }
```

The routine initializes the package. Its arguments are the same as those to main itself. It is an error to not call before any other getpar calls. The routines are deactivated with a call to This routine releases memory, and allows the option mentioned below to happen. The individual parameters are obtained with calls to and (must par) is identical to except that it terminates the program with an error message if the particular parameter is not specified. The following description of also applies to has three parameters. The first is a character string which specifies the external name of the parameter. It can be (practically speaking) of any length. The second parameter is a character string which specifies the type of variable the parameter is. Currently the following types are understood:

"d"	integer
"f"	float
"F"	double
"s"	character string
"b"	boolean (integer)
"vd"	integer vector
"vf"	float vector
"vF"	double vector

The *type* parameter can be used to indicate the maximum number of elements allowable in vectors. For example *type="vf[4]"* or *type="vf(4)"* would cause *getpar* tp modify no more than 4 elements of the vector, regardless of how many elements the user specifies. If no limit is specified, a limit of 10 is quietly enforced. The last parameter is a pointer to the type of variable indicated by does not modify this variable if no occurence of the parameter is found. Hence, a default value can be assigned before the call to returns 1 the parameter was found, and 0 if not. For vectors, returns the number of elements found. The parameters on the command line can occur in any order, and any number of times. In the case of multiple specifications, the last one is used. Any parameters that are not requested by or are ignored. An example of specifying parameters for the above program is: a.out dx=0.123 nx=300 title="sample title" dy=0.456 noboo x=1.0,4x2.0,2x5.12 Each specifcation is of the form No imbedded blanks are allowed on either side of the equals ("=") sign. Character strings with blanks or tabs are delimited with single (’) or double (") quotes. The only exceptions to the rule are boolean variables which are specifed as or to indicate true or false. Boolean variables may also be specified as integers with the form In the above example, true values for are specified as either or and false values as either or If is given then the returned value is The value for vector is given as a list separated by commas (,). No imbedded blanks are allowed in the list. Repetition factors (2x and 4x in the above example) can be used to specify repeated values. Several additional features are also available. At any point on the command line, the parameter can be given. This will cause to look in the file for additional parameters. Several arguments can be given on the command line. The search order is left to right. Consequently, any parameters given after the will override their values given in Also, the environment (if allowed, see NOENV option below) is searched first. Thus parameters on the command line and in par files override parameters set in the environment. The format of the parameters in the par file follow the same rules as the command line. Several specifications separated by while space, can occur on a given line, and their can be any number of lines. A ’#’ symbol in the position where a name would normally occur, indicates that the rest of the line is a comment, and is consequently ignored. The specification is also allow in the file, however recursions are limited in depth (current limit is 4). As a concession to the traditional switch passing method, a parameter of the form a.out -abc is available to the calling program as a character string with the call getpar("SWITCH","s",&sw); where in the example above, the string would be Parameters in the shell environment can be set (unset) with the C-shell commands: setenv name value unsetenv name The parameters in the environment can be printed with the command printenv Five additional parameters allow for input checking, and program interogation. The call to will terminate the program if this parameter is given. Each call to or will cause the name, type, and value of the variable to be listed on If is given, the listing is put in the file This option is useful for interogating a program as to what it wants for input. will list all input parameters that are found. This option is useful for debugging input data, and determining where a particular parameter is coming from in multiple specifications. If is given, the listing is put in the file will disallow any parameters to be obtained from the environment. It may occur on the command line, in a par file, or in the environment itself. will cause to print the name of the parameter before starting to search for it. This provides a quick method of determining which subroutine call is at fault, when a program dies in the getpar package. The routines are in the library and may be loaded with getlist(3), getarg(3) Be careful that is correctly specified, when dealing with floats and doubles. If a double pointer is used with *type="f"*, The lowest 32 bits of the mantissa will not be set correctly. If a float pointer and *type="F"* are used, the next element in memory will be clobbered. If the last combination is used in vector mode, you will get garbage back. If a program appears to be behaving differently for identical input parameters, make sure that some defaulted parameters are not sneaking in via the environment. The NOENV option may be of some use in this case. If you believe back door parameters are a bad practice, then set NOENV in your login shell. The routines produce error messages of the type:

```
****** ERROR program[getpar]: ******
          error message
```

Hopefully the error message is diagnostic of the trouble. If you are debugging a program try the VERBOSE option to see which call generated the problem. Errors in calls to are often reported as calls to Robert W. Clayton, Seismological Laboratory, Caltech, Pasadena, CA 91125