SeismicHandler

K. Stammler

27 April 1992

Contents

1 General Information

SeismicHandler (SH) is a tool for analysing digital seismograms. It can be used for the analysis of earthquake records as well as for examining seismogram sections in refraction seismology. The program was developed during the work at my PhD at the SZGRF in Erlangen. Excluding graphic interfaces it now consists of roughly 40,000 lines of source code, written in ANSI C. The current version was developed on ATARI ST/TT and has been exported to a MicroVAX and to Sun computers, so the portability should be guaranteed.

The program uses dynamic storage allocation. The length of the traces SH can hold in memory is limited only by the RAM size of the computer. The maximum number of traces in memory is currently limited to 300. Trace editing functions such as spike removal, baseline correction or polynomial interpolation are available.

Although SH will run on a Tektronix terminal, it can more conveniently be used on a window system. Currently implemented window graphics interfaces are X–Window (tested on VAX and SUN), VWS (VAX/VMS window system) and GEM (ATARI). On a window system you have at least two different windows, one dialog window for entering commands and a second for graphic output. SH is able to handle up to seven windows, but you will rarely use more than three. It is, for example, convenient to open a separate window for particle motion diagrams.

The user interface of SH is a command language designed specifically for processing seismic data. Command lines are typed in interactively or may be read from a file (command procedure). Dialog boxes, pop–up or pull–down menues have not been implemented due to the incompatibility of different graphics interfaces.

There is a help text available for each SH command. It can be requested interactively by the `help` command. If you create new commands you can update the help library just by adding a help text file to the help directory.

2 The Command Line

The command interpreter parses each command line by processing two steps:

1. split up the command line in words using blanks, semicolons and slashes as terminators. Words are separated either by one or more blanks or by a single semicolon or by a single semicolon and additional blanks or by a slash which may be preceded by blanks. Two consecutive semicolons or two semicolons with only blanks in between denote an empty word.

2. translation of each word. A description of the translation process is given in section 3.

All words which were separated by a slash ("/"), are identified as qualifiers (see below). Qualifiers are treated separately and do not count as parameters. Besides these qualifiers the first word is regarded as the command verb, the others as parameters numbered from 1 to N. N ranges from zero to fifteen. Empty words (see above) result in an empty parameter. Note that the first parameter can be separated from the verb by a semicolon as well. That means if you want to pass an empty first parameter and a second parameter `par` to a command verb `verb`, you have to type `verb;;par` and *not* `verb;par`, where `par` is regarded as the first parameter.

Qualifiers are most often used as switches, using their presence as one state of the switch, their absence as the other. Such simple qualifiers `/SimpleQual` consist of the slash ("/") and the name of the qualifier. However, on some commands you may pass valued qualifiers `/ValuedQual=value`. The value `value` is appended to the valued qualifier with a "="-character in between. There is a maximum number

of 5 qualifiers per command permitted. A complicated example of a command line is

```
verb par1;par2 ;; par4/SimpleQual par5 par6  /ValuedQual=val.
```

The exclamation character "!" is regarded as the end of the command line. Any characters following it are ignored and may be used as a comment on the command line. A command line beginning with the exclamation character is equivalent to an empty line.

3 Translation of Expressions

As it was mentioned in the previous section, the command parser translates each word of the given command line if possible. The translation process is applied if the word starts with one of the following non–alphanumeric characters: `" # $ % _ ^ |`

The "`|`"–characters are used for concatenation of subexpressions, which may be translated itself before. The construction `|<sub1>|<sub2>|<sub3>|`, for example, results in a concatenation of the three subexpressions `<sub1>`, `<sub2>` and `<sub3>`. There is a maximum of 10 subexpressions which can be concatenated within one command line.

All others of the previously listed characters require either an indexed or a non–indexed name to follow. A non–indexed name is just a string of alphanumeric characters `<name>`, an indexed name consists besides the name an index string in paranthesis `<name>(<index>)`. Now follows a list of valid translatable expressions:

`"<name>` Expression is replaced by the current value of the local or global variable `<name>`. See section 8.3 for detailed information about variables in SH. `<name>` must be a defined variable name. See command `sdef`.

`#<name>` If `<name>` is a number between 1 and 15 the expression is replaced by the value of the `<name>`–th parameter passed to the current command procedure (see section 8). If `<name>` equals the string `params` it is replaced by the total number of parameters passed to the command procedure. Any other expression accesses qualifiers of the command line which called the current procedure. For example, the word `#example` is replaced by

- the string `_EXISTSNOT_` if the qualifier `/example` was not specified.
- the string `_NOVALUE_` if the qualifier `/example` was specified without value.
- the value `qualvalue` if a valued qualifier `/example=qualvalue` was specified.

`$<name>` This evaluates internal variables. An example is `$dsptrcs` which returns the current number of traces on the display. A complete list of all internal variables is given in section 9.

`%<file>(<line>)` Returns the `<line>`–th line of the text file `<file>`. If the paranthesses and `<line>` are omitted the first line of `<file>` is returned. A value of 0 for `<line>` returns the total number of lines in `<file>`. If the file extension is not specified it is assumed to be `.stx` which is the default extension for SH text output files (`echo` command). Please make sure that the text file doesn't contain lines longer than 132 characters. Otherwise the line counting won't be correct.

`^<info>(<trace>)` Returns the value of the info entry `<info>` of the `<trace>`–th trace on display (counting from bottum up). If the trace number and

the parantheses are omitted the first trace is accessed. As an example, the expression `^delta(3)` is replaced by the sample distance of the third trace on display. For detailed information about info entries see section 4.

`_<info>(<start>:<end>)` Such an expression is translated only if it is found in place of a trace list parameter. It selects all traces which have an `<info>`–value between `<start>` and `<end>`. A more detailed explanation is given in section 5.

In an indexed name both subexpressions (`<name>` and `<index>`) itself can be of the above (non–indexed) type. As an example, the expression `%#1("cnt)` is replaced by the line number `cnt` of a text file, whose name is passed as the first parameter to the current command procedure.

4 Information Entries

Usually there is a lot of information about a seismogram besides the sample data itself. Some examples are the *length* of the trace, the *sample distance*, the recording *station* and *component* name. All these information values and many more can be stored in the q–files, the preferred data format of SH. The two most convenient issues of SH concerning the q–file format are (i) the accessibility for reading and writing of each of these information values through SH by descriptive names and (ii) the possibility of defining your own information entries for the q–files (including their names).

To use the information entries you don't have to know much about the q–file format, but you should know a little about how SH accesses them. The most important thing about information entries in q–files is, that they have a type (called Q–TYPE) and a number (called Q–NUMBER). For your convenience there is a name assigned to each information entry (called INFONAME), which SH can translate into Q–TYPE and Q–NUMBER to identify the information uniquely. But there is a problem concerning the execution speed. If SH would read the Q-file header on each read–access to any of the information entries, this would slow down the program drastically. A write–access is even worse, because SH would have to rewrite the whole Q–header file. For this reason SH stores frequently used information entries for each trace in memory. That means that every INFONAME does not only point to a Q–TYPE and a Q–NUMBER, but as well to a type (called SH–TYPE) and an index (called SH–INDEX) in memory. The SH–INDEX is different from q–number, because the SH–INDEX controls whether or not the information entry is stored in memory (see below). So you can change the set of information entries hold in memory by changing their SH-INDEX numbers without changing the Q–NUMBERs. It is necessary to distinct between Q–TYPE and SH–TYPE, because there exist more SH–TYPEs than Q–TYPEs. That means, some SH–TYPEs do not exist on q–file and are converted to an existing Q–TYPE to store it on q–file. For example the SH–TYPE TIME is stored as a STRING Q–TYPE.

On a read–access to an information value SH translates the given INFONAME to SH–TYPE and SH–INDEX. By the value of SH–INDEX SH can tell whether the information is hold in memory. If this is the case, SH takes the information found in memory, it doesn't touch the information in the q–file. If the information is not found in memory, SH determines Q–TYPE and Q–NUMBER and reads it from q–file.

Any write–access (command `set`) to information values changes only values in memory by default. If the information is not found in memory, SH doesn't change anything. Only on explicit request (command `set/file`) SH changes information entries on file (and in memory if it is there as well).

Some information entries do not exist on q–file but are hold in memory. These entries are created when the trace is read or generated in any other way. This is useful, for example, for entries like minimum and maximum amplitude of a trace or any information concerning the display, like time origin, vertical position, display attributes, normalization factors and so on.

As already mentioned, the SH–INDEX controls the storage type of the information entry. There were mentioned three storage types of entries. These are the frequently used information entries stored in q–file *and* hold in memory (called AUTO–LOAD entries, because they are loaded automatically into memory when a trace is read in), the rather slow accessible entries stored in q–files only (called FILE–ONLY entries) and the temporary information entries, which are hold in memory only (called MEM–ONLY entries). For each SH–TYPE there is a different but fixed number of entries (called MAXMEM) which can be hold in memory. This number includes the MEM–ONLY entries which are predefined and cannot (or at least should not) be changed. Therefore remains a smaller number (called MAXAUTO) of AUTO–LOAD entries which are definable by the user. In detail the index structure is like this: all entries with an SH–INDEX ranging from 0 to MAXAUTO-1 are AUTO–LOAD entries, all entries from MAXAUTO to MAXMEM-1 are MEM–ONLY entries, all entries greater or equal to MAXMEM are FILE–ONLY entries. Note that SH supports only SH–INDEX numbers smaller than a number MAXFILE. The values of the numbers MAXAUTO, MAXMEM and MAXFILE for each entry type and all currently defined information entries including their SH–INDEX and Q–NUMBER can be listed by the SH command `entry list outfile.txt`. Then a file `outfile.txt` is created and typed on the current output window of SH.

Remember that the SH–INDEX is an SH–internal number and does not appear in q–files in any way. For the identification of an entry, only the Q–NUMBER is decisive. This allows you to change the set of entries which are used as AUTO–LOAD from one SH session to another.

Now follows a complete list of all available entry types (all are valid as SH–TYPEs and most of them are valid Q–TYPEs)

LONG SH–TYPE and Q–TYPE. 32–bit signed integer value.

INTEGER SH–TYPE and Q–TYPE. 16–bit signed integer value.

BYTE SH–TYPE only, its converted to INTEGER type on q–files. 8–bit signed integer value.

REAL SH–TYPE and Q–TYPE. Real number in floating point or exponential format.

STRING SH–TYPE and Q–TYPE. Character string containing any printable character except tilde ("~").

CHAR SH–TYPE and Q–TYPE. Single printable character (no tilde "~" permitted).

TIME SH–TYPE only, its converted to STRING type on q–files. Absolute time specification, containing date and time.
Example: 23–JUN–1989_23:30:00.000

FLAG SH–TYPE only, its converted to CHAR type on q–files. Two–valued entry, possible values are `yes` and `no`.

Since you can create your own information entries, there are predefined only the entries which are internally used by SH. You should not change the SH–INDEX or the Q–NUMBER of any of these predefined entries, even if you can. If you do so, you

will slow down the program (in the best case) or crash it (the worst case). This is a complete list of the predefined entries:

LENGTH AUTO–LOAD, SH–TYPE and Q–TYPE LONG, Q–NUMBER 1. Length of trace in number of samples.

ALLOC MEM–ONLY, SH–TYPE LONG. Size of allocated memory for the trace in units of samples. Usually this is the same as LENGTH.

DSPFST MEM–ONLY, SH–TYPE LONG. Index number of first sample inside the current display window. Controlled by commands `STW` and `DTW`.

DSPCNT MEM–ONLY, SH–TYPE LONG. Number of samples inside the current display window. Controlled by commands `STW` and `DTW`.

RECNO MEM–ONLY, SH–TYPE INTEGER. If the trace is read from a q–file, this entry contains the position number of the trace inside the file, otherwise the entry value is zero.

ATTRIB MEM–ONLY, SH–TYPE INTEGER. Number of the display attribute block for the trace. A display attribute block controls the output attributes of traces like colour, line width and line style *and* the output attributes of text like colour, size, font and text effects. For details about attribute blocks see section 10. The number of available attribute blocks depends on the implemented graphics package. The default attribute block is 0.

REDUCTION MEM–ONLY, SH–TYPE INTEGER. Reduction factor for trace plotting. Reduces number of samples on display and increases output speed. If, for example, the reduction factor is 3, every third point of the trace is plotted. The default value of the reduction is 1, that means every point is plotted.

DELTA AUTO–LOAD, SH–TYPE and Q–TYPE REAL, Q–NUMBER 0. Usually this is the sample distance in seconds.

MAXVAL MEM–ONLY, SH–TYPE REAL. Value of the maximum sample of the whole trace. Determined automatically when the trace is read and after every trace manipulation.

MINVAL MEM–ONLY, SH–TYPE REAL. Value of the minimum sample of the whole trace. Determined automatically when the trace is read and after every trace manipulation.

NORM MEM–ONLY, SH–TYPE REAL. Normalization factor. Determined automatically before each redraw, depending on the selected normalisation mode (see command `norm`).

ZOOM MEM–ONLY, SH–TYPE REAL. Zoom factor entered by user via command `zoom`. Default value is 1. The total amplification factor of the trace is the product of `norm` and `zoom`.

T-ORIGIN MEM–ONLY, SH–TYPE REAL. Horizontal position of trace. Given in the same units as `delta`. Controlled by commands `shift` and `al`, `beam` and others.

S-ORIGIN MEM–ONLY, SH–TYPE REAL. Vertical position of trace. Determined automatically before each redraw. Computation algorithm can be modifed with `yinfo` command.

WEIGHT MEM–ONLY, SH–TYPE REAL. Weight of trace when used in a `sum` command. Default value is 1.

COMMENT AUTO–LOAD, SH–TYPE and Q–TYPE STRING, Q–NUMBER 0. Comment line on trace.

STATION AUTO–LOAD, SH–TYPE and Q–TYPE STRING, Q–NUMBER 1. Station code of recording station.

FILE MEM–ONLY, SH–TYPE STRING. Name of input file of trace. If the trace is generated in SH the string is copied from a parent trace if possible, otherwise it remains empty.

COMP AUTO–LOAD, SH–TYPE and Q–TYPE CHAR, Q–NUMBER 0. Component of recording station.

START AUTO–LOAD, SH–TYPE TIME, Q–TYPE string, Q–NUMBER 21. Start date and time of record.

MODIF MEM–ONLY, SH–TYPE FLAG. If the trace is modified since read in from q–file, this flag is set to `yes`, otherwise its `no`.

FROMQ MEM–ONLY, SH–TYPE FLAG. If the trace is read from a q–file this flag is set to `yes`, otherwise its `no`.

When you define your own entries, you should make sure that

- the Q–NUMBER is not used by another already defined entry (this is not checked by SH !)
- the Q–NUMBER is not greater or equal to MAXFILE
- the SH–INDEX is not used by another already defined entry (this is checked)
- the SH–INDEX is not that of a MEM–ONLY entry, that means it should not be inside the range from MAXAUTO to MAXMEM-1
- the SH–INDEX is not greater or equal to MAXFILE

Usually additional entries are defined within the startup command file of SH `shstrtup`. To get a complete list of the currently defined entries, their SH–INDEX numbers and their Q–NUMBERs, enter the command `entry list outfile.txt`, which creates a file `outfile.txt` and types it on the current output window. To define entries use the command `entry define`. See `help entry` for detailed information about the `entry` command.

5 Trace Addressing

Many commands require a list of traces as input parameter. Then you will usually specify a number of traces of the current display. The traces are addressed by their position number inside the display window. The positions are counted from the bottom up to the top, starting at 1. By default, the position numbers are displayed on the left side of each trace, where the name of the recording station and the component are given as well. But this trace labelling can be changed (command `trctxt`) and so the trace numbering is not necessarily visible. Let's use the command `del` as an example. You may use the following list expressions in place of any parameter of type `trace list` (the parameter type is always given in the help text to each command). The delete command `del` takes only one parameter which

is the list of traces to be deleted. Suppose you want to delete the bottom trace only. This is done by `del 1`. After execution the display is redrawn automatically (if the display redraw is enabled) without the deleted first trace. The bottom trace which had position number 2 before the delete command, now got the position number 1. It is possible to specify a set of traces by a list of position numbers, separated by commas. For example, to delete the first, third and fifth trace, the command is `del 1,3,5`. A block of consecutive numbers can be specified by the first and last number of the block, separated by a hyphen. So the command `del 1-6` deletes the first 6 traces on the display. Blocks and lists may be combined which means `del 1-3,7-9,14-16` is a valid command and deletes the traces with the position numbers 1, 2, 3, 7, 8, 9, 14, 15, 16. Please keep in mind, that any such expression must not contain blanks, because otherwise the command parser would regard the expression as more than one parameter and the parameter passing won't be correct. To select all traces of the display window, use the expression `all`, which means `del all` deletes all displayed traces.

It is possible to change the order of the traces on display by the `display` command. For example, `display 3 1` puts trace number 3 to the bottom by changing it's position to 1. Also valid is `display 7-9 3`, which takes the traces at the positions 7 to 9 and inserts them at positions 3 to 5.

Sometimes it is useful to remove traces from the display and keep them in memory. Then the traces can be redisplayed later without reading them from file again (which takes time and you need to remember the filename) or without computing them again, if the traces are not read from file directly. The command is `hide <list>`. `<list>` may be any trace list parameter as described above. Then the traces specified in `<list>` are hidden, they are not displayed any more. They are redisplayed by the `display` command. But since the traces are not visible, they don't have a position number which is usually needed for addressing. Therefore a special expression `h:all` is defined which includes all hidden traces. The command `display h:all 3` displays all hidden traces starting at position number 3. The second parameter may be omitted. It is then assumed to be 1. With this command all of the hidden traces are redisplayed. To redisplay only a subset, you need an expression which is explained in the following.

There exists a possibility to select a subset of all traces in memory which match a condition given for a specified info entry. A range for this info entry is selected and all traces which have an info entry within this range are put on the list. The general syntax of such an expression is `_<info>(<start>:<end>)`. `<info>` is the name of the info entry. `<start>` and `<end>` specify the limits of the value range. As an example, let's assume that you want to display only traces which have an epicentral distance between 30° and 50°. First all traces must be hidden, using `hide all`. Then redisplay all traces matching the given condition `display _distance(30:50)`. Traces which don't have a `distance`–value at all, are not put on the list as well as all traces whose distance value is out of the specified range. If the lower bound `<start>` is not specified, all traces with a distance smaller than 50° are selected (`display _distance(:50)`). The same holds for the upper bound `<end>`, `display _distance(30:)` displays traces with a distance larger than 30°. The expression `_distance(30)` selects traces which have a distance value of exactly 30°. For real–valued entries this is usually senseless, but for other entry types like STRING or CHAR or TIME it may be useful. So the command `display _comp(z)` displays only z–components. But notice that SH converts the command line to uppercase letters by default. This is important for string and character comparisons. You can negate the selection conditions by putting a "~"–character after the info entry name. This means, the expression `_distance~(30:50)` selects all traces whose distance value is *not* in the range between 30° and 50°. Some instructive examples of commands using list expressions are given below.

`del 4-6,8,11,15-20`
Delete traces number 4, 5, 6, 8, 11, 15, 16, 17, 18, 19, 20.

`hide all`
Hide all traces.

`display h:all`
Display all hidden traces starting at position 1.

`sum _azimuth(0:180)`
Sum traces whose azimuth value is between 0 and 180°.

`del _comp~(z)`
Delete all traces which are not z–components.

`zoom/rel _station(bji) 2`
Enlarge the amplitude of all traces of station BJI by a factor of 2.

`hide _magnitude(:5.8)`
Hide all traces with a magnitude smaller than 5.8.

6 Trace Filtering

Application of filters is a very important step in analysing seismic traces. It is performed in almost any case of data processing. For this reason the related commands are explained here in more detail in addition to the interactive help texts. SH knows three different kind of filters: FFT–filters, recursive filters and tabulated filters. Besides these SH can perform operations related to filtering, like Hilbert–transformation, attenuation and computation of minimum delay signals from given signals or autocorrelations.

6.1 FFT Filters

This is the most common way of trace filtering. A copy of the trace is transformed into frequency domain using the FFT (Fast Fourier Transformation) algorithm. If the number of samples is not a power of 2, zeroes are appended to the trace automatically. In the frequency domain the trace is multiplied by the filter function which is given as poles and zeroes (In detail, the trace is multiplied with the first zero, then divided by the first pole, then multiplied with the second zero, then divided by the second pole and so on until all zeroes and all poles are used). After this process the trace is transformed back into time domain and displayed on screen (if the redraw is enabled). SH stores the currently used filter (or a cascade of filters) in memory. That means, before applying the filter operation you have to read in an FFT filter file, containing poles and zeroes and a normalization factor. This is a text file which you can create with any text editor or with utility programs. An example of such a filter file is given here:

```
! file WWSSN_SP.FLF
!      ============
!
! WWSSN-LP FFT filter (including GRF restitution)
! H(s) = ( P(s,h0,w0) / P(s,h1,w1) ) * ( w2*w2 / P(s,h2,w2) )
! where P(s,h,w) := s*s + 2*h*w*s + w*w
! h0 = 0.707,  t0 = 2*pi/w0 = 20.0
! h1 = 0.67,   t1 = 2*pi/w1 = 1.05
! h2 = 0.55,   t2 = 2*pi/w2 = 0.75
!
1357913578
```

```
1
70.18385353
2
(-0.2221106,-0.2221777)
(-0.2221106,0.2221777)
4
(-4.009271,-4.442279)
(-4.009271,4.442279)
(-4.607669,-6.996659)
(-4.607669,6.996659)
```

At the beginning of the file may be (better: should be) comment lines. Comment lines start with an exclamation sign "`!`" (no preceding blanks !). The first line after the comments is a magic number and must be `1357913578`. This identifies the file to be an SH filter file. The next line is another ID number, specifying the filter type. For FFT filter files this is `1`. The next line contains a normalization constant which is multiplied to the filtered trace. Then follows, again in a separate line, the number of zeroes of the filter. In the example file two zeroes are given. A pair of complex conjugated zeroes counts as two zeroes and both of them must be specified in consecutive, separate lines. The real and imaginary part are separated by a comma and enclosed in parantheses. Even real numbers must be entered in this format. The poles are given similarly. First the number of poles, then the complex poles in separate lines. Please note that blank lines are not permitted at any place in the file.

The given filter is a simulation filter for a WWSSN–SP seismomemter which needs a velocity–proportional record of an STS–1 instrument as input. The filter file `WWSSN_SP.FLF` is read into memory by the command `fili f wwssn_sp`. The default extension `.FLF` may be omitted. The filter files are searched in the current directory and in the filter library (see section 11 about default paths in SH). The `f` as second parameter denotes that the file contains an FFT filter. It is possible to specify more than one filter file on this command. For example, the input `fili f f1 f2 f3` reads in the three FFT files `F1.FLF`, `F2.FLF` and `F3.FLF`. The filters are applied to the trace one after the other (filter cascade). If the qualifier `/compress` is specified, all filters are concatenated to a single filter and zero–valued poles and zeroes are shortened if possible (non–zero values are not shortened, I'm sorry for this). Once the filter (cascade) is read in, it remains in memory until another `fili`–command replaces it. Each filter process uses the filter(s) read in by the most recent `fili`–command. To apply the filter to the first three traces on display, type in `filter f 1-3`. Again, the `f`–parameter denotes an FFT filter process. After execution of this command, three new traces appear on top of the display containing the filter output. FFT filters are quite slow if the input traces are very long (several ten thousands of samples). It is possible to set a time window for the filter process. If you want to filter only the time window between 50 and 350 seconds (relative to the time axis), enter `filter f 1-3 50 350`. By default, that means without parameters 3 and 4, the whole trace is filtered (not only the part inside the display window).

6.2 Recursive Filters

Recursive filters have the advantage that they are faster on long traces than FFT filters. The reason is, that output samples are computed as linear combinations of already existing samples of the input and the output trace. Thus the computation time grows proportional to the number of samples N, while the FFT filters grow proportional to $N \log N$. On the other hand, recursive filters have some deficiencies

as well. These are mainly:

- Usually it takes more time to determine the recursive filter coefficients than the poles and zeroes for a given transfer function. Particularly if the poles and zeroes are given, an FFT filter can be written down immediately, while a recursive filter needs a considerable amount of brainwork.
- The filter coefficients depend on the sample rate. Such a recursive filter can be applied only to traces of a fixed sample rate.
- On very long–period filters, where many samples are involved to compute a new output sample, recursive filters tend to numerical instabilities.
- Output traces of recursive filters usually have high–amplitude numerical noise at their beginning.

As SH supports both filter types it is up to the user to decide which filter is to be preferred in his application.

The formula used in the recursive filter operations is this:

$$r_n = a_0 f_n + a_1 f_{n-1} + \ldots + a_k f_{n-k} - b_1 r_{n-1} - b_2 r_{n-2} - \ldots - b_l r_{n-l}$$

where the $a_i, i = 0, \ldots, k$ and the $b_j, j = 1, \ldots, l$ are the filter coefficients. The input trace is given by the f's and the output trace by the r's. This means, the current output sample r_n is determined by the current input sample f_n, k previous input samples and l previous output samples. The first k output samples access input samples with indices smaller than zero, which are not known. These are assumed to be zero. This is the reason for the numerical noise at the beginning of the output trace.

An SH recursive filter file can contain one or more recursive filters. The whole filter file is read in by the command `fili`. An example file `WWSSN_SP.FLR` of a WWSSN-SP filter is given here:

```
! file WWSSN_SP.FLR
!      ============
!
! WWSSN-SP recursive filter
!
1357913578
3
0.05
1.0
3
1.011167
-1.999877
0.9889559
3
1.224691
-1.954563
0.8207457
@
3
0.05
1.0
3
4.5180295e-2
```

```
9.0360589e-2
4.5180295e-2
3
1.278993
-1.909639
0.8113681
```

It again needs as input velocity–proportional records of an STS–1 instrument. The first lines beginning with "`!`" are comment lines and are ignored by SH (but not by the human reader of the file !). The first line after the comments must be the magic number `1357913578`. The next line contains the ID number of recursive filters and is `3`. This is followed by the sample distance (in sec) of the coefficients. SH stores this value and permits the user to apply this filter only to traces of this sample rate. The next line is a normalization number and is usually 1. Then the number of a–coefficients is specified, which is `3` in the example file. That means, this number is followed by 3 coefficients, namely a_0, a_1 and a_2. The same holds for the b–coefficients. The example file gives three b's, b_0, b_1 and b_2. Please note that in the filter file a b_0 must be specified which does not appear in the above formula (in fact, this is the coefficient of the r_n on the left side of the equation). SH eliminates b_0 after reading the file by dividing all $a_i, i = 0, \ldots, k$ and $b_j, j = 1, \ldots, l$ by b_0. After the coefficient b_2, the first filter in the example file ends. The next line contains the separation character "`@`", indicating that another recursive filter is appended. The second filter starts with the ID number `3` for recursive filters (there is no more magic number). Then, again, follows the sample distance, the normalization, the a–coefficients and the b–coefficients. This example file contains a cascade of two recursive filters. You can use up to 5 filters in one file. The command `fili r wwssn_sp` reads the whole filter file into memory. The default extension `.FLR` may be omitted. Recursive filters of a previous `fili`–command are replaced by this cascade (FFT filters are not affected by this command). The command `filter r all` applies the filter cascade to all of the traces on display. The result traces are appended to the top of the display. As for FFT filters you can specify a time window by optional parameters (number 3 and 4). These denote the lower and upper bound of the window (in sec) relative to the time axis.

6.3 Tabulated Filters

Sometimes a transfer function may be given as a tabulated function instead of poles and zeroes or it is desirable to create an acausal filter in the frequency domain. In such cases you can use the tabulated filters of SH. This is a text file containing the tabulated values of amplitude and phase transfer functions. These tabulated values may be non–equidistant. The filter process is performed similar to the FFT filters. A copy of the input trace in transformed to the frequency domain and each frequency sample is multiplied by an interpolated value of the tabulated filter. The amplitude and phase function of the filter are both linearly interpolated. The filtered trace is transformed back to the time domain and displayed on screen.

A simple example file `TRAPEZ.FLT` of a tabulated filter is listed below.

```
! file TRAPEZ.FLT
!      ============
!
! simple tabulated filter
!
1357913578
2
```

```
6
0.0   0.0   0.0
0.02  0.0   0.0
0.03  1.0   0.0
0.3   1.0   0.0
0.5   0.0   0.0
100.0 0.0   0.0
```

The first lines with a "`!`"–character in the first column are comments. The first line after the comments is the well–known magic number `1357913578`. It is followed by the ID number `2` of tabulated filters. The next line contains the number of tabulated filter values (here `6`). Then follows one line for each point in the frequency domain. Every line consists of three numbers. First the frequency in Hz, then the amplitude function and at last the phase function. The example shows a trapezoidal amplitude function and a zero phase function. The transfer function is flat (amplitude 1) between 0.03 Hz and 0.3 Hz. Designing such a filter you should make sure that the tabulated function covers at least the area in frequency domain which is used by the input trace, that is from zero to the nyquist frequency. If you fail to do so the filter command will abort with an error message. The filter file is read in with the command `fili t trapez`. The default extension `.FLT` may be omitted. The filter operation is done by `filter t <list>`, where `<list>` denotes any trace list. As for the other filters you can specify a time window in optional parameters number 3 and 4.

6.4 Special Filters

SH knows some other filter operations which are used sometimes. The Hilbert transformation is one of them. It is applied with the command `filter h <list>`, where `<list>` may be any trace list like `4-6` or `2` or `all`. As for any other filter commands, the input traces remain unchanged and the new output traces are appended to the top of the window. You can restrict this operation to a time window if you specify the lower and upper bound (in sec, relative to the time axis) as parameter numbers 3 and 4, respectively.

Another option is the attenuation of a trace by an attenuation operator

$$A(\omega) = e^{-\frac{1}{2}\omega t^* + i\frac{\omega}{\pi}\ln\frac{\omega}{\omega_N}}$$

where t^* is the attenuation parameter and ω_N is the Nyquist frequency. t^* is passed to the command as parameter number 5. If you don't want to set a time window for the operation, then you have to enter empty parameters 3 and 4. The command to attenuate the first three traces on display by a t^* of 0.5 s is `filter a 1-3;;;0.5`.

One option related to filter operations remains to be mentioned in this section. It is the computation of a minimum delay signal. Input is either an arbitrary time signal or an autocorrelation. If a time signal is given, the command is `filter m <list>;;;<shift>`, for an autocorrelation it is `filter c <list>;;;<shift>`. `<list>` is a trace list like `4-6` or `all` and `<shift>` is a time shift in s by which the output trace is shifted to the right. If you specify 0 for the `<shift>` parameter then the signal on the output trace starts exactly at the beginning of the trace which is not very convenient in most cases. Of course, it is possible to enter a time window in parameters 3 and 4 as in the other filter commands.

7 Scaling of Amplitudes

If more than one trace is in the display window of SH one has to consider, how to relate the individual trace amplitudes. If the records of a 3–component seismometer or traces of a station array are displayed, everyone prefers to have the same amplification factors for each trace. This makes sure that the true amplitude ratios are shown. In other cases where several records of the same event from globally distributed stations are displayed or where filtered and unfiltered waveforms are to be compared, it is often more convenient to have the scaling in such a way that each trace is plotted with the same amplitude. Also it may be desirable sometimes to amplify a subset of the displayed traces in amplitude. SH can manage each of these problems, there exist quite a few commands to manipulate the trace amplitudes. This section describes how to use this subset of commands.

It is useful to know a bit about how SH determines the actual display amplitudes. Each trace has two temporary info entries (MEM-ONLY type, see section 4), called NORM and ZOOM. The display amplitude results as the product of the seismogram amplitude (this is given by the sample values) and these two real numbers. The NORM entry is determined by SH before each redraw, depending on the current active normalization mode. The ZOOM entry is set by the user. Thus exist three different ways to manipulate the trace amplitudes:

1. Changing the normalization mode using the `norm` command. This tells SH how to determine the normalization factor (NORM entry) for each trace.

2. Changing the zoom factor (ZOOM entry) via `zoom` command.

3. Changing the sample values of the trace (commands `trcfct` and `unit`). This is somewhat problematic if this is done only for reasons of the trace display, because it really changes your traces. If you sum these traces at a later time, they may have the wrong weights. If you write the traces to a file, the amplitude changes are saved as well. Therefore you should prefer the commands `norm` and `zoom`. The entries NORM and ZOOM are temporary and are not saved in an output file. Also, these entries don't have any influence on operations with the traces, they affect the display only.

point 1: The normalization mode is changed by the `norm` command. This command accepts only one parameter, which must be one out of a set of five short strings:

`af` The normalization factor is the same for each trace and is determined as half of the reciprocal value of the maximum sample (without sign) of all displayed traces on their total length (not only inside the displayed window) using the info entries MAXVAL and MINVAL. That means, all traces are normalized to the maximum value on all traces within their full length.

`aw` Here the normalization factor also is the same for each trace, but the maximum value is determined only within the displayed window. The traces are normalized to the maximum value on all traces within the displayed window. Since the determination of the maximum value is done before each redraw and since the info entries MAXVAL and MINVAL cannot be used here, this normalization mode may slow down the program considerably, if long traces are to be searched.

`sf` The normalization is determined for each trace separately as half of the reciprocal value of the maximum sample (without sign) on it's full length (not only inside the displayed window) using the info entries MAXVAL and MINVAL. That means, all traces have the same display amplitude, if the ZOOM–factors are identical.

`sw` Normalization is again determined separately, but now inside the current display window, resulting in equal display amplitudes for traces with the same zoom factor. Since the determination of the maximum value is done before each redraw and since the info entries MAXVAL and MINVAL cannot be used here, this normalization mode may slow down the program considerably, if long traces are to be searched.

`c` All normalization factors are set to 1. This is useful for applications, where different plots must have the same amplitude scale. In all other modes the normalization depends on the sample amplitudes of the traces on display. If you make plots of two different data sets you cannot compare the amplitudes between the plots. This problem is solved, if you use the `c`–mode and if you always set the same zoom factor and if you have always the same number of traces on display. But after switching to the `c`–mode, you have to be aware that the display amplitudes may look like zero (if the sample amplitudes are very small) or may be immense (for big sample amplitudes). In any case you have to find out appropriate zoom factors by yourself.

The modes `af` and `sf` sometimes confuse users, if large amplitudes exist outside the display window. In this case the display amplitudes of some or all traces are very small. This can be checked either by deleting the time window or by switching to the modes `aw` or `sw`. But keep in mind that the latter modes can slow down the program if you deal with long traces. The default mode (if not changed in the setup file) is `af`.

point 2: The zoom factor is changed via the `zoom` command. The syntax is `zoom <list> <factor>`. Since there is a `<list>` parameter, it is possible to scale the traces independently. `<factor>` is copied to the ZOOM entry of the specified traces. It remains there until it is explicitely changed. Since the value is copied to the entry, a command `zoom 1 2` doesn't change anything, if the first trace is already zoomed by a factor of 2. But it is possible to enter relative factors as well. This is done by `zoom/rel 1 2`, which magnifies the display amplitude of the first trace by a factor of 2, no matter what the zoom factor currently is. Traces which are created by SH operations and which are appended to the top of the display, get a default zoom factor which is usually 1. This means, if all traces on the display have a zoom factor of 5 and a new trace is created, this appears to be relatively small, because of the default zoom factor of 1. However, with the command `zoom/default <list> <factor>` you can change the default zoom factor (and the zoom factor of the traces in `<list>`).

point 3: If all else fails (or seems to be too inconvenient) the samples itself can be multiplied by a factor. But as it is mentioned above, this might result in problems in future operations on such traces. You have always to keep in mind that these traces do not have the original sample values any more. A subset `<list>` of traces can be multiplied by a number `<r>`, using the command `trcfct <list> mul <r>`. On the display amplitudes this will have the same effect as `zoom/rel <list> <r>`. Therefore this command isn't really necessary for reasons of display. More convenient is the `unit` command. It determines the absolute maximum (without sign) of a given list of traces and within a given time window. This maximum is set to 1 and all other samples of the specified traces are normalized with respect to this absolute maximum (the sign remains untouched). A typical application is, if there are two or more three–component sets of records on the screen, with large amplitude differences between the sets. With `unit` each set can get the maximum sample amplitude of 1. This way the sets can be compared without loosing information about the relative amplitudes within a set. The syntax of the `unit` command is `unit <list> <lo> <hi>`. `<list>` specifies the set of traces to be normalized together.

<lo> and <hi> contain the lower and upper bound of the time window in s (relative to the time axis) where to look for the absolute maximum. If these parameters are omitted, the full traces are used.

8 Command Procedures

The ability of SH to process command procedures is one of it's most important features. Most of SH's internal commands operate on a rather low level. Therefore SH is very flexible and can be used for many different and quite special purposes. On the other hand, more complex operations consist of several basic instructions and require a considerable amount of keyboard input. For the convenience of the user it is therefore often necessary to combine these low–level instructions and create more elaborate commands. Thus the user interface is optimized for the solution of particular data processing problems.

A command procedure is an editable text file. Each line of this file holds a single SH command (and/or comments). The commands are processed sequentially until the `return` command is found. This terminates the execution of the current procedure and returns to the parent command level which is either the interactive level or another command procedure. The end of file also terminates execution but it creates an error message and returns to the interactive level in any case. The command procedure is called by the name of the command file without extension which is assumed to be `.SHC`.

8.1 A first example

The usage of command procedures will be explained here by developing an example file which will get more complex step by step. The purpose of this command file is to perform a 3–dimensional rotation from the recording Z,N,E–coordinate system to the local ray system of the P–wave, called L,Q,T–system. This rotation needs two angles, the azimuth and the angle of incidence. For the first example let's assume that there exists a q–file named `q_exm` which contains a three component record with the components Z, N and E at the file positions 1, 2 and 3, respectively. The azimuth and angle of incidence are known. Let their values be 207.2° and 19.5°, respectively. The following procedure `rotex1` reads in the traces, rotates them and writes the result to an output q-file, named `q_out`.

```
! file rotex1.shc
!
! version 1 of the 3-dim rotation procedure
! K. Stammler, 15-Apr-92

del all             ! delete possibly existing traces ...
dtw                 ! ... and time window

read q_exm 1-3      ! read in Z,N,E components
rot 1-3 207.2 19.5  ! rotate, create three new traces
write q_out 4-6     ! write result traces to output q-file

return              ! return to parent level
```

After entering the command `rotex1` this procedure is executed if SH can find the command file. SH looks for the file `rotex1.shc` in the current directory and in a common command directory. Of course, the q–file `q_exm` has to be in your current directory, otherwise you have to specify the directory explicitely.

8.2 Command Parameters

In practice the above command procedure is not very useful, because the azimuth and angle of incidence are given as fixed values in the file. These angles are usually different for each event. Instead of changing the text each time, it is much more convenient to pass these values as parameters to the procedure. In the command procedure the parameter number `N` is accessed by the expression `#N`. So the rotation command `rot 1-3 207.2 19.5` of the procedure should be replaced by `rot 1-3 #1 #2`. Then the procedure works fine as long as the user passes the azimuth as the first parameter and the angle of incidence as the second. But imagine the case that the user can't remember the order of the parameters or doesn't know the parameters at all. In particular for more complex procedures with five or ten parameters this problem would be even worse. Guessing parameters is very tedious and annoying. Therefore a command is implemented which prompts the user for the parameters if he wants to. Besides that this command assigns default values to parameters which are left empty in the calling command line. For this reason the command is called `default`. The first parameter of the `default`–command specifies the number of the parameter, the second contains the default value. If this is empty, no default value is assigned. All following parameters are used as a prompt text if the user is to be prompted for input. If the command procedure is called without any parameters (not even an empty parameter) all parameters of the procedure are prompted using the given prompt text of the `default`–command. If the user specifies at least one parameter (even if it is empty), no parameter is prompted. Instead, every empty or not specified parameter gets its default value from the `default`–command. The updated version of the rotation procedure now looks like this:

```
! file rotex2.shc
!
! version 2 of the 3-dim rotation procedure
! K. Stammler, 15-Apr-92

default 1 ;;    input q-file        ! input file
default 2 ;;    azimuth             ! no default for azimuth
default 3 0.    angle of incidence  ! default 0 for incidence
default 4 q_out output q-file       ! output file

del all             ! delete possibly existing traces ...
dtw                 ! ... and time window

read #1 1-3         ! read in Z,N,E components
rot 1-3 #2 #3       ! rotate, create three new traces
write #4 4-6        ! write result traces to output q-file

return              ! return to parent level
```

The command line `rotex2 q_ex 207.2 19.5 q_out` is now equivalent to the first version of the procedure `rotex1`. If you enter `rotex2 q_ex 207.2`, the angle of incidence defaults to 0 and the output file defaults to `q_out` without any prompting. The plain command `rotex2` without parameters let SH prompt you for all of the parameters in the specified order, using the prompts given in the `default` command. At the third and fourth parameter, additionally to the prompt text, a default value is offered. You can accept it just by hitting the return key, otherwise you have to enter another value. You should use the `default`–command on every parameter in order to supply the user with information texts about the parameters. If you do so, the command procedure behaves exactly like an internal command.

8.3 Variables in SH

The recent version of the rotation procedure is not really a big improvement for the user. He still needs to know the rotation angles for each event and has to pass it to the command procedure. SH can determine these rotation angles with the command `mdir`. But variables are needed to store the results of `mdir` and to pass them to `rot`.

The concept of variables in SH is similar to other programming languages. There exist global and local variables. Local variables are visible only inside the command procedure where they were defined. The local variables are deleted when the procedure is terminated. Global variables are visible in any command level, that means in all command procedures and in the interactive level. In order to keep a good programming style there shouldn't be defined too many global variables (as in any other language). Usually the four predefined global variables are sufficient. Their names are `g1`, `g2`, `g3` and `ret`. These variables are needed only to store return values of command procedures.

All variables must be defined. Any access to an undefined variable results in an error message. The definition command is `sdef` (symbol definition). The first parameter is the name of the variable to be defined. An optional second parameter specifies its initial value. By default the defined variables are local. For a global definition the qualifier `/global` is required. SH variables have no type. That means in a variable you can store any information, like integers, floating point numbers, strings and others. SH does not check any type information, so you have to be aware of what you are doing. In fact, any values are stored as strings and are converted internally if necessary. To assign a value to a variable, it has to be passed to a command which returns an output value. In this case a `&`–character has to be placed in front of the variable name, to indicate a write access. A read access to a variable is made by a preceding `"`–character. The command parser assumes that any word in the command line beginning with a double quote is a defined variable and replaces the expression by its current value.

Now follows version 3 of the rotation procedure which let the user select a time window to determine the rotation angles via `mdir`–command.

```
! file rotex3.shc
!
! version 3 of the 3-dim rotation procedure
! K. Stammler, 16-Apr-92

default 1 ;;    input q-file        ! input file
default 2 q_out output q-file       ! output file

sdef azim                 ! define azimuth variable, no init
sdef inci                 ! define angle of incidence, no init

del all                   ! delete possibly existing traces ...
dtw                       ! ... and time window

read #1 1-3               ! read in Z,N,E components
echo select time window   ! message to the user
mdir 1-3 *;;&azim &inci   ! determine angles from time window
rot 1-3 "azim "inci       ! rotate, create three new traces
write #2 4-6              ! write result traces to output q-file

return                    ! return to parent level
```

In earlier versions of SH `mdir` won't work properly inside a command procedures if the time window is user–selected (`*`–parameter). In this case two additional variables holding begin and end of the time window must be defined. Their values are assigned using the command `time` twice (example: `time;;&start`). The window bounds must be passed to the command `mdir` by these variables.

8.4 The Command `calc`

An important command for assigning values to variables is `calc`. It performs simple numerical computations as well as text and time manipulations. The general syntax is:

```
calc <type> <outvar> = <operand1> [<op> <operand2>] [<p>]
```

This fixed structure allows only one operation per command line. Here the same syntax rules hold as for any other command. That means in particular that the "`=`" character and the operator `<op>` are treated as ordinary parameters which must be separated by blanks (or semicolons) from the preceding and following expressions. `<type>` specifies the type of the operation. This is necessary, because all variables can have any type and SH needs to know whether there is to compute an integer addition or a floating point addition or a string concatenation, for example. `<outfile>` gives the name of the variable (preceding the `&` character) where to store the result of the operation. If only one operand is specified the instruction performs a plain assignment of the value of `<operand1>` to `<outvar>`. `<p>` is an additional parameter which is used only in a few special operations.

Valid operators for the type "`i`" (integer) are:

`+`	Adds the two operands.
`-`	Subtracts `<operand2>` from `<operand1>`.
`*`	Multiplies the two operands.
`div`	Divides `<operand1>` by `<operand2>` without remainder.
`mod`	Remainder of the division `<operand1>` by `<operand2>`.

Valid operators for the type "`r`" (real) are:

`+`	Adds the two operands.
`-`	Subtracts `<operand2>` from `<operand1>`.
`*`	Multiplies the two operands.
`div`	Divides `<operand1>` by `<operand2>`.
`abs`	Removes the sign of `<operand1>`. No second operand.
`arctan2`	Computes the arc tangens of `<operand1>/<operand2>` in degrees. Works correct even if `<operand2>` is equal or close to zero.
`power`	Takes `<operand1>` to the `<operand2>`–th power.

Additionally there exist a number of single argument functions. The argument is specified by `<operand1>`, `<operand2>` must be empty. All trigonometric functions (and their inverse) work with degrees as input (output). A complete list of the functions is `sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, exp, log, ln, sqrt`.

Valid operators for the type "`s`" (string) are:

`+` Concatenation of `<operand1>` and `<operand2>` with a blank in between.

`parse` Extracts the `<operand2>`–th word from `<operand1>`. Separation characters are blanks and semicolons. `<operand2>` must be integer.

`extract` Extracts `<p>` characters from the string `<operand1>` starting at position `<operand2>`. The first character of `<operand1>` has number 1 (not 0). `<operand2>` and `<p>` must be integer.

Valid operators for the type "`t`" (time) are:

`tdiff` Computes the difference of two absolute time values `<operand1>` and `<operand2>`. The difference is given in seconds and stored as a floating point number.

`tadd` Adds `<operand2>` seconds to the absolute time value `<operand1>`. `<operand2>` must be a floating point number, negative numbers are accepted. The result is again an absolute time value.

`cnv_julian` Converts the day number `<operand2>` of the year `<operand1>` into day and month. The two output integers are stored in `<outvar>` separated by a blank. Both operands must be integer (in an appropriate range).

`make_time` Converts an absolute time string in numeric format (examples: `"30,-12,85,5,30,20,5"` or `"85/12/30/5/30/20/5"`) to the standard format of absolute time (`"30-DEC-1985 5:30:20.005"`).

Examples:

```
calc i &cnt = "cnt + 1       ! increments counter cnt
calc r &num = "a * "b        ! multiplies two floating points
calc r &num = "x sqrt        ! takes square root of x
calc s &str = "str parse 1   ! keeps first word only in str
calc s &str = "x extract 5 3 ! extracts chars 5 to 7 from x
calc t &dif = ^p-onset(3) tdiff ^start(3)
                             ! computes time offset of P onset
                             ! relative to start of trace 3
```

With this command the rotation procedure can run completely without user interaction. Supposition is, however, that the P–wave onset is inserted to the q–file header. There exists a predefined info entry, called `p-onset`. It should contain the absolute time of the P onset. The user still has to select the P–wave times for all his events like in the previous versions of the procedure, but now he has to do it only once. Moreover, this process is decoupled from the rotation procedure and the gathered information can be used for other applications as well. The insertion of the onset time is done by two commands if the event is already on screen. The command `time &g1` let the user select the onset time by graphic cursor and stores

the result in the (global) variable `g1` which is predefined. Of course, you can use any other variable as well. The second command is `set/file all p-onset "g1`, which inserts the determined time to the headers of all traces on display. For details see command `set`. Another info entry used in the command procedure is the start time of the trace. This is also an absolute time, the name of this (predefined) entry is `start`. Now follows the automated version 4 of the rotation procedure. It writes the L–, Q– and T–components to separate files.

```
! file rotex4.shc
!
! version 4 of the 3-dim rotation procedure
! K. Stammler, 20-Apr-92

default 1 ;;    input q-file          ! input file
default 2 q_out output q-file prefix  ! output file prefix
default 3 2.    window width (sec)    ! time window for
                                      ! determining angles

sdef azim                 ! azimuth
sdef inci                 ! angle of incidence
sdef start                ! start of time window (relative time)
sdef end                  ! end of time window (relative time)

del all                   ! delete possibly existing traces ...
dtw                       ! ... and time window

read #1 1-3               ! read in Z,N,E components
calc t &start = ^p-onset(1) tdiff ^start(1)
                          ! determine relative time of P-onset
calc r &end = "start + #3 ! get end of time window
mdir 1-3 "start "end &azim &inci
                          ! determine angles in computed window
rot 1-3 "azim "inci       ! rotate, create three new traces
write |#2|_l| 4           ! write L-component to L-file
write |#2|_q| 5           ! write Q-component to Q-file
write |#2|_t| 6           ! write T-component to T-file

return                    ! return to parent level
```

8.5 Loops

Version 4 of the rotation procedure is already well developed. It enables the user to process several 3–component records. But suppose he has 100 events stored on q–files. He would have to type in 100 similar command lines. Even if he would write a command procedure calling `rotex4` 100 times, he would have to insert the 100 file names to the procedure text. It is more convenient to store the 100 file names in a separate file (this may be created with a directory command redirected to a file) and process all files in a command loop.

For loop structures SH provides two commands, `goto` and `if`. `goto` has the syntax

```
goto <label>
```

This command requests SH to jump to the specified label `<label>`. `<label>` is a text string with a colon ":" at the end. SH looks for this label inside the current

command procedure. If the qualifier `/forward` is specified it starts looking at the current position, otherwise it rewinds the command file before searching. A label is any text string at the beginning of a command line that ends with a colon. Labels are ignored in ordinary execution of command procedures. They only serve as jump addresses. Label lines must not contain any executable commands.

The command `if` performs the conditional execution of an instruction. It's syntax is

```
if  <e1> <cmp> <e2>  <cmd> [<label>]
```

The sequence `<e1> <cmd> <e2>` is a compare operation between two expressions `<e1>` and `<e2>` with a compare operator `<cmp>`. If the result of this comparison is true, the instruction `<cmd>` is executed. Only two SH instructions are permitted in place of `<cmd>`, namely `return` and `goto`. `return` terminates the current command procedure and `goto` jumps to a specified label `<label>`.

The compare operator `<cmp>` specifies the actual comparison and the type of the expressions `<e1>` and `<e2>`. Valid operators are:

`eqi` integer comparison, `<e1>` is equal to `<e2>`

`nei` integer comparison, `<e1>` is not equal to `<e2>`

`lei` integer comparison, `<e1>` is less or equal `<e2>`

`lti` integer comparison, `<e1>` is less than `<e2>`

`gei` integer comparison, `<e1>` is greater or equal `<e2>`

`gti` integer comparison, `<e1>` is greater than `<e2>`

`eqr` float comparison, `<e1>` is equal to `<e2>`

`ner` float comparison, `<e1>` is not equal to `<e2>`

`ler` float comparison, `<e1>` is less or equal `<e2>`

`ltr` float comparison, `<e1>` is less than `<e2>`

`ger` float comparison, `<e1>` is greater or equal `<e2>`

`gtr` float comparison, `<e1>` is greater than `<e2>`

`eqs` string comparison, `<e1>` is equal to `<e2>`

`nes` string comparison, `<e1>` is not equal to `<e2>`

With this supplement it is possible to write a command procedure to apply `rotex4` to all q–files listed in a list file.

```
! file rotex5.shc
!
! version 5 of the 3-dim rotation procedure
! K. Stammler, 21-Apr-92

default 1 ;;      q-file list ! name of list file
default 2 1       first file  ! first file to process
default 3 %#1(0)  last file   ! last file, default is last line
default 4 q_out   output file ! output file prefix
default 5 2.      time window ! width of window
```

```
sdef cnt #2                    ! file counter init. to start line
sdef qfile                     ! name of current q-file

nr                             ! switch off redraw, incr. speed
loop_start:                    ! start of loop (only label)
   if  "cnt gti #3  goto/forward loop_exit:
                               ! exit if counter exceeds last file
   calc s &qfile = %#1("cnt)   ! get current q-file from list
   echo processing file "qfile! message to the user
   rotex4 "qfile #4 #5         ! call rotex4
   calc i &cnt = "cnt + 1      ! increment counter
goto loop_start:               ! repeat loop
loop_exit:                     ! loop exit
rd                             ! switch on redraw again

return                         ! return to parent level
```

This example file also demonstrates nesting of command procedures, because this procedure, `rotex5`, calls another procedure, `rotex4`. Nesting is permitted up to a level of nine calls, which is very likely sufficient in all applications. In the above example the `nr` command switches off the automatic redraw, which increases execution speed. But don't forget to switch it on again (`rd`).

8.6 Execution Flags

Execution flags control some details in the processing of a command procedure (some affect the interactive level as well). Examples are the suppression of error messages, abortion of procedures on errors and tracing through a procedure. Such features can be switched on and off by the `switch` command. It's syntax is `switch <flag> <on/off>`. `<flag>` is the name of the execution flag and `<on/off>` is either `on` or `off`. A list of all flags is given below:

`cmderrstop` X–flag. Controls whether the command procedure is aborted on errors. If it is switched on, SH returns to the interactive level after displaying the error message. Usually this option is switched on. After such an error the command procedure should be corrected. Be careful in switching off this flag, because you may get lot of messages of subsequent errors. In the worst case you may create an infinite loop in the command procedure and you will have to abort SH in a very crude way. Default is `on`.

`sherrstop` A–flag (Abort). Controls whether SH is terminated after an error occurred. This is useful only if SH is executed in batch mode. Default is `off`.

`verify` V–flag. Controls whether a verification of each command is printed before it is executed. With this option the translation of the command lines can be checked, since the command is printed after the translation process. Default is `off`.

`echo` E–flag. Controls whether all commands are echoed (without translation) before execution. Default is `off`.

`step` T–flag (Trace). Controls whether SH prompts the user for keyboard input (`<Return>`–key) before each command. Useful in combination with V–flag. Default is `off`.

`protocol` P–flag. Controls whether the interactive commands of the user are logged in the protocol file of SH. Default is `on`.

`capcnv` C–flag. Controls whether each input character is converted to uppercase. Default is `on`.

`noerrmsg` Q–flag (Quiet). If this switch is on, no error messages (and no warning bell) is printed. Useful in combination with X–flag if possibly occurring errors are handled by the command procedure. Default is `off`.

`chatty` I–flag (Info). Some commands give an explaining text if this flag is switched on (like command `sum`). This info text may be switched off if the internal structure of the command procedure should be hidden from the user. Default is `on`.

By default, the switches are changed only locally within the current command procedure. The flags are reset when the procedure terminates. To change flags globally, the qualifier `/global` must be entered on the `switch` command. Then the switches remain in the specified state until they are changed again by a `switch/global` command. Besides the `switch` command there exists another method to change the flags. When a command procedure is called, valued qualifiers `/flags`, `/flags+` or `/flags-` may be applied. The value of the qualifier is a string consisting of one or more flag characters. The flag characters are given in the above list. `/flags` sets all specified flags to `on`, all others to `off`. `/flags+` sets all specified flags to `on`, all others remain unchanged. `/flags-` set all specified flags to `off`, all others remain unchanged. The flags are changed only locally within the called command procedure, unless the `/global` qualifier is also specified. With the `/global` qualifier the changes affect all child levels of the called procedure.

8.7 Debugging Tools

Most of the execution flags mentioned in the last section are not very important for most users. But two of them are very useful if a command procedure contains a mistake which must be detected. With the V–flag all translated commands can be listed in the dialog window. Usually the last command before the error message caused the error. In some more complicated cases the whole procedure must be traced step by step. Additionally to the V–flag, the T–flag lets the user check each command carefully by prompting for keyboard input (`<Return>`–key) before each command. If you want to debug the command procedure `bugproc`, you have to enter `bugproc/flags+=v`. Possibly existing command parameters can be passed as usual: `bugproc/flags+=v p1 p2`. This call enables the command verification. If you want to switch on trace mode as well, the command is `bugproc/flags+=vt`. The flags are set only in `bugproc`, if `bugproc` calls another command procedure, this is processed without verifying and trace mode. To make SH tracing all child levels as well, type in `bugproc/flags+=vt/global`. If the error is detected, it may be tedious to continue tracing until the procedure terminates. An input of "`@`" terminates tracing (and the command procedure). Of course, the V– and T–flags may be set by the `switch` command as well, but then you have to change the text of the command procedure.

In general when a command procedured is aborted by an error a *status report file* is created in the SH scratch directory. The file name is a concatenation of the SH Session ID and the string `ERR.STX`. The Session ID always starts with `SH` or `SH$` or `SH_`, followed by a random number. The random number is necessary to make each SH session ID unique on a multi–tasking operating system. All scratch files of SH start with the Session ID string to avoid interference of different SH sessions

with each other. The same holds for the status report files. A typical name for a status file on VMS is `SH$2354_ERR.STX`. It contains error number, error message and the calling chain (all levels) of the command procedures including the line numbers where the error occurred. Also given are the execution flags, all defined local (named *symbols set 0*) and global (named *symbols set 1*) variables and all parameters and qualifiers passed to the procedure. This information is very helpful for debugging command procedures.

9 Internal Variables

Internal variables are a bit different from the local and global variables that the user can define with `sdef`. They are a fixed set and are defined internally by SH. Also they do not contain information about traces as the trace info entries do. Internal variables are read–only for the user. Some of them reflect SH status parameters, like `$dsptrcs` or `$tottrcs`, others contain fixed values, like all the character variables (`$blank`, `$dollar`, ...). All names start with a "`$`"–character and are replaced by the command interpreter by it's current value. A complete list is:

`$dsptrcs`	Number of traces in the display window.
`$tottrcs`	Total number of traces in memory.
`$status`	Return status of the last executed command. Zero means successful completion, any other value is an error number.
`$systime`	Current system date and time string. This is not yet implemented in all operating systems.
`$version`	Returns current version of SH.
`$dsp_x`	Returns the lower bound of the current time window (set by command `stw`).
`$dsp_xmax`	Returns the upper bound of the current time window (set by command `stw`).
`$dsp_w`	Returns the width of the current time window (set by command `stw`).
`$dsp_y`	Returns the lower bound of the current vertical window (y–window, set by command `syw`).
`$dsp_ymax`	Returns the upper bound of the current vertical window (y–window, set by command `syw`).
`$dsp_h`	Returns the width of the current vertical window (y–window, set by command `syw`).
`$titlestyle`	Returns style block number of the title text lines. For additional information about attribute blocks, see section 10.
`$trcinfostyle`	Returns style block number of the trace info text
`$zerotrcstyle`	Returns style block number of the line attributes of zero traces
`$timeaxisstyle`	Returns style block number of the line style of the time axis and the label text
`$markstyle`	Returns style block number of the marker lines

`$x` This is a special variable, pointing to the currently drawn trace on the display. It is useful only in connection with the command `trctxt`.

Additionally there exist character values which return special characters. These variables may be indexed, like info entries. The index number in parantheses is interpreted as a repeat counter. For example, the expression `$blank(10)` returns a string of ten blanks. You need the `$blank` variable if you want to pass a string parameter which contains blanks. If you would specify the blanks in the parameter directly, SH wouldn't regard the string as a single parameter and the parameters would be passed incorrectly. The correct parameter specification of an example parameter "`text with blanks`" is:

```
|text|$blank|with|$blank|blanks|
```

Besides `$blank` other special characters are available. The names are self–explaining, here follows only a list of their names: `$exclamation, $quotes, $dollar, $percent, $hat, $bar, $slash, $number`. You can create any ASCII character with a `$hexchar??`–expression. The two question marks stand for the two digits of the character's ASCII code (hexadecimal). For example, two consecutive horizontal tabulators are created by `$hexchar09(2)`.

10 Output Attributes

Every output, text or lines, to the graphics window of SH is controlled by attribute blocks. An attribute block stores information, how thick a line is drawn, which line style and which colour is used. Also it determines the text attributes like character height, font and colour. There are several attribute blocks available, the exact number depends on the actual graphics interface (at least 10). Most of these blocks are for free use, but some are reserved for special output items.

To change an entry in an attribute block you have to use the command `fct setstyle <block> <item> <value>`. `<block>` specifies the attribute block number, `<item>` the item to be changed and `<value>` it's new value. Valid items are:

`linewidth` Set line width. `<value>` specifies the line width in pixels.

`linestyle` Set line style. `<value>` is an integer number specifying the line style. Default style is 0 (continuous line). All other styles are dashed or dotted lines.

`color` Set line colour. `<value>` is an expression defining the colour. In X–Window and VWS there must be specified three real numbers between 0 and 1, separated by commans (no blanks). These numbers are the red, green and blue fractions of the output colour. Usually, there are colour files `RED.STX`, `YELLOW.STX GREEN.STX` and so on in the `globals`–directory of SH.

`charsize` Set the size of characters. `<value>` specifies the character height in units of window height. For example, the labelling of the time axis has usually the size 0.02.

`font` Set character font. `<value>` contains the font number.

`wrmode` Set writing mode. `<value>` is either `replace` or `xor`. `replace` is the default mode.

As mentioned above, some attribute blocks are reserved for special use. The actual numbers of these blocks may be different in each implementation and they may change in future versions of SH. Therefore are internal variables defined, which contain the attribute block numbers of various output items. A complete list of these internal variables can be found in section 9. An example is `$markstyle`, which is the style block number of the vertical trace markers (lines) and their labelling (text). Usually the marker lines are red (on coloured screens). If you want to change the line width to five pixels, you have to enter `fct setstyle $markstyle linewidth 5`. All following markers are then thick red lines. Of course, you can change the colour to blue: `fct setstyle $markstyle color %blue`. In this command the colour expression is read from the file `BLUE.STX` in the `globals` directory of SH. Such colour files exist for the most common colours. It is possible to add other colour files if necessary.

The traces on display do not have fixed attribute blocks. By default, they use block number 0 for output. All traces have an info entry, called `attrib`, which contains the current block number. So each trace can use a different attribute block if it is necessary and if enough blocks are available. The entry value is changed by the `set` command, as any other info entry. To change the colour of the traces 4-6 to red, you need two commands. First set an attribute block (for example number 1) to red colour: `fct setstyle 1 color %red`. Then assign this block to the selected traces: `set 4-6 attrib 1`. After the next redraw (command `rd`), these traces will be displayed in red colour. At the end of this section some examples are given.

`fct setstyle $timeaxisstyle linewidth 3`
Change line width of time axis to thicker lines.

`fct setstyle $markstyle wrmode xor`
Change trace markers to XOR–mode. Now the markers can be removed if they are drawn a second time at the same position.

`fct setstyle 3 color %yellow`
Set colour of attribute block number 3 to yellow.

`set all attrib 3`
Use attribute block 3 for all traces on display.

`set 1,4,6 attrib 3`
Use attribute block 3 for traces 1, 4 and 6.

`set _magnitude(7.0:)  attrib 3`
Use attribute block 3 for all traces with a magnitude larger than 7.0.

11 SH Paths and Input Files

Many operations like filter input or calling a command procedure require input files. These input files are searched in the current directory and, if they are not found there, in special default directories. SH knows several separate directories for help files, for command procedures, filters and others. The actual paths are usually defined in the startup file of SH. It is very unlikely that they have to be changed after SH is installed properly, nevertheless it is useful to explain the path commands here. The command syntax is `fct path <name> <dir>`. `<name>` specifies the name of the directory to be set and `<dir>` contains the actual path. Examples for `<path>` are `/home/sh/filter/` in UNIX and `disk1:[main.sh.command]` in VMS. It follows a list of valid directory names and a short description of their content.

`scratch` Directory for scratch output files like hardcopy files, protocol files, error files and so on. This directory must not be changed within a command procedure, because otherwise the status flags and local parameters of the

parent command level cannot be restored properly. This is due to the fact, that when a command procedure is called, the status, local variables and parameters of the parent level are saved to a temporary file in the scratch directory (`.SSV`–files). If the scratch directory is changed within this procedure this file cannot be found any more after the procedure has terminated.

`help` Contains all help files about built–in commands and command procedures. Help files have the extension `.HLP`. New files can be added to this directory if new command procedures are to be documented.

`command` Directory of common command procedures. Procedures in this directory are available to all users of SH.

`globals` Default directory for access to input text files, that means if the file `text.stx` in an expression `%text("cnt)` cannot be found in the current directory, it is searched in this directory.

`filter` Filter files of any type (FFT, recursive and tabulated) in this directory are available to every user of SH.

`errors` Contains the error messages of SH. The corresponding text of an error number 1820, for example, is found in the file `ERR_1800.MSG` in line 20.

`inputs` Directory for input data files like travel time tables (`.TTT`–files) of various phases.

12 Travel Times

There is a utility routine implemented in SH to read travel times of various phases from travel time tables. These travel times may be used to mark the theoretical arrival times in the seismograms. The travel time tables are assumed to be in the `input` directory of SH. This default directory may be changed by the command `fct tt_table <path>`, where `<path>` contains the actual directory path of the travel time tables. Of course, the tables have to be in a special format.

Each seismic phase is stored in a separate file, called `.TTT`–file. The name of the file is given by the name of the phase (for example `P.TTT` for P–phases or `PKP.TTT` for PKP–phases). Since many operating systems are not case–sensitive, phases like pP or PcP need a special convention. All lowercase letters in the phase names are converted to uppercase with a preceding "V"–character. That means, the travel times of the phase PcP are read from a file `PVCP.TTT`, pP from `VPP.TTT` and so on. In extreme cases like Pdiff (`PVDVIVFVF.TTT`) this is a bit clumsy, but this doesn't happen too often. The `.TTT`–files itself are ASCII files of the following format:

```
! travel time table for PP-waves
!
! created by Ray Buland's program TTIMES
TTT
distance bounds
27.0 180.0
depth steps
15 0.0 50.0 100.0 150.0 200.0 250.0 300.0 350.0 ...
 27.0 000000 000000 000000 391.70 388.42 385.42 382.77 380.48 ...
 28.0 000000 409.94 406.33 402.81 399.53 396.53 393.87 391.57 ...
 29.0 426.71 421.05 417.44 413.92 410.64 407.63 404.96 402.65 ...
```

```
 30.0 437.82 432.16 428.55 425.02 421.73 418.72 416.04 413.71 ...
 31.0 448.93 443.27 439.65 436.12 432.82 429.79 427.10 424.75 ...
    :
    :
```

The first lines beginning with "`!`" are ignored by SH. The first line after the comments contains an identification string `TTT` (uppercase letters). The next line after this is ignored and is used for comments. The following line specifies the distance bounds in which valid travel time information is available in the file. Then follows again a comment line. The next information is the number N of depth steps (`15` in the example file). Then the depth values (in km) of all N depths must be specified. After this header information the travel time data are listed. Each line contains travel times of one epicentral distance. The distance value in degrees is the first number in the line. It is followed by N travel times in seconds, one for each depth step defined above. Travel time of illegal combinations of depth and distance are given as zeroes.

If the travel time for given distance and depth is requested SH opens the file and reads the header information. If distance or depth are out of range the routine is aborted and an error message is displayed. Otherwise SH scans all lines until the distance found is greater than the requested distance (it doesn't check EOF, so the distance bounds of the header must be definitely correct). It takes this and the previous line and interpolates the travel times linearly in distance and depth. The resulting time is returned. To call the travel time utility, enter the command `call travel <phase> <distance> <depth> <result>`. `<phase>` specifies the seismic phase. If the name contains lowercase letters, you have to use the "V"–convention or you turn off the automatic case conversion (then you have to enter uppercase commands and keywords). `<distance>` and `<depth>` are the distance and depth of the event in degrees and km, respectively. `<result>` is the name of the output variable (with a preceding "`&`"–character) where to store the resulting travel time. Examples:

`call travel p 98.0 33.0 &tt`
After this call the variable `tt` contains the resulting travel time of P (`tt` must be a defined variable). It may be used in subsequent commands as input. To inspect the variable use the `echo` command as for any other expression: `echo "tt`.

`call travel vpp 50.0 0.0 &tt`
Computation of pP travel time. An equivalent command sequence is given below.

```
switch capcnv off
CALL TRAVEL pP 50.0 0.0 &TT
SWITCH CAPCNV ON
```

Returns also pP travel time in variable `tt`.

A short command procedure `markphase.shc` explains how theoretical arrival times of an arbitrary phase are marked on a seismogram. Supposition is that the trace is read from a q–file with given informations about origin time (entry `origin`), distance (entry `distance`) and depth (entry `depth`). If this is not the case, additional parameters to the command procedure must be defined, supplying it with the requested information.

```
! file MARKPHASE.SHC
!
! marks phase on a selected trace
! K. Stammler, 27-Apr-92
!
default 1 1    trace number        ! which trace to mark
```

```
switch capcnv off
DEFAULT 2 P    phase name          ! case sensitive prompt
SWITCH CAPCNV ON

sdef tt                            ! output travel time

call travel #2 ^distance(#1) ^depth(#1) &tt
                                   ! compute travel time
calc t &tt = ^origin(#1) tadd "tt  ! get absolute arrival time
mark/abs/label=#2 #1 "tt           ! mark time position

return
```

Important is that the second parameter is only case–sensitive if the procedure is called without parameters and all parameters are prompted. A call like `markphase 1 PcP` won't work, because first the command parser converts all letters to uppercase and then passes the parameters to the procedure `markphase`. In this case you have to use the "V"–convention: `markphase 1 pvcp`.

13 Event Locations and Beams

If data from a station array are available, SH is able to determine azimuth and slowness of a selected phase. Conversely, it computes beam traces if azimuth and slowness are given. These operations need to know at which stations the individual seismograms are recorded and where the stations are located. The first information is easy to get, because there exists a predefined info entry `station` specifying the recording station. The second part, the location of the stations is independend from individual seismograms and is therefore not available in info entries. SH uses a special text file which lists station information for many different stations. This file is accessed if one of the above operations (commands `locate` and `beam`) is performed. If it is not found, the operation is aborted. To tell SH where to find the location file, use the command `fct locfile <file>`. `<file>` specifies the complete filename including path and extension. An example file `statloc.dat` shows the format of such a file.

```
GRA1 +49.6918877 +11.2217202 1 -21.923 +37.862   Graefenb...
GRB1 +49.3913475 +11.6519526 1  +9.085  +4.308   Graefenb...
GRC1 +48.9961681 +11.5213504 1  -0.775 -39.662   Graefenb...
GRA2 +49.6552079 +11.3594439 1 -11.985 +33.668   Graefenb...
GRA3 +49.7622038 +11.3186951 1 -14.815 +45.618   Graefenb...
GRA4 +49.5654029 +11.4358711 1  -6.575 +23.668   Graefenb...
GRB2 +49.2709252 +11.6699661 1 +10.145  -9.162   Graefenb...
GRB3 +49.3435419 +11.8059826 1 +20.165  -1.112   Graefenb...
GRB4 +49.4689373 +11.5608463 1  +2.445 +12.858   Graefenb...
GRB5 +49.1121310 +11.6767332 1 +10.785 -26.972   Graefenb...
GRC2 +48.8675675 +11.3755426 1 -11.595 -53.902   Graefenb...
GRC3 +48.8901739 +11.5858216 1  +3.805 -51.472   Graefenb...
GRC4 +49.0867465 +11.5262720 1  -0.355 -29.602   Graefenb...
WET  +49.14      +12.88      0   0.      0.      FRG, Wetzell
BFO  +48.3        +8.3       0   0.      0.      FRG, Black F...
HAM  +53.46       +9.92      0   0.      0.      FRG, Hamburg
CLZ  +51.8       +10.4       0   0.      0.      FRG, Clausthal
BRL  +52.2       +13.5       0   0.      0.      FRG, Berlin
FUR  +48.16      +11.28      0   0.      0.      FRG, Fuerst...
```

```
TNS  +50.22       +8.45      0   0.      0.      FRG, Taunus
```

Each line contains one station, starting with the name of the station (usually in uppercase letters), the latitude and the longitude in degrees. The next number is an array ID code (integer), followed by two floating point numbers specifying the relative array positions of the stations in km. All following text in the line may be used as a comment. Please make sure that no line exceeds the length of 132 characters. By the array ID code the stations can be grouped to arrays. Each array must have a unique array number (is `1` for the GRF–array in the example file). Stations which do not belong to an array must get the array code 0. Each array station must also have relative coordinates (these are the last two numbers in the line). Stations with an array code of 0 may have zero values for the relative coordinates. The command `locate` uses the relative positioning rather than latitude and longitude if all traces passed are recorded at the same station array (which means all have the same non–zero array code). This can be prevented by specifying the `/noarray`–qualifier on the `locate` command.

To compute azimuth and slowness of a phase, `locate` needs it's travel time differences at each station. If you just enter `locate` (or `locate/noarray`) you have to pick the phase at each station by graphic cursor (exit the selection by pressing the "`E`"–key). `locate` uses the plain time differences between the picks, it does not account for time–shifted traces. So please make sure that all traces are correctly positioned in time with reference to each other. If all traces have the same start time, this can be achieved by resetting all time shifts to zero by the command `set all t-origin 0`. If the traces have different start times you need the additional command `shift all time_al` to align the traces in time. The results of the computation are displayed on screen or are copied to output variables if specified. The general syntax is `locate <list> <azim> <inci> <azim-err> <inci-err>`. The first parameter `<list>` is not used in this computation mode and should be empty. All other parameters specify output variables to store results.

The `beam` command acts inversely and applies time shifts to the traces, determined from the station locations and from given slowness and azimuth. The calling syntax is `beam <list> <azimuth> <slowness>`. `<list>` specifies which traces are shifted (typically `all`) and `<azimuth>` and `<slowness>` are the (back–)azimuth in degrees and slowness in s/deg. `beam` applies relative shifts, like the `shift` command. If you apply `beam` twice you get twice the time shift at each trace. To align traces in time before the `beam` command, apply the commands given above. To obtain a beam trace you have to sum the `beam`ed traces. A typical command sequence is:

```
set all t-origin 0
shift all time_al
beam all 330.0 5.4
sum all
```

It creates the beam trace on top of the display. If this beam process is applied for many different slownesses at a fixed azimuth (or reversely for a set of azimuths at a fixed slowness) you get many summation traces which are comparable to a vespagram. This is what the library routine `vespa` does. The parameters are `vespa <slo-start> <slo-end> <slo-step> <azimuth> <power>`. `<slo-start>` and `<slo-end>` specify the slowness interval, the step size is given by `<slo-step>`. For each slowness step one beam trace is computed. The fixed azimuth is given in `<azimuth>`. `<power>` determines the order of the N-th Root Process applied in the summation. The `vespa` command uses all traces on display for beaming. After execution the input traces are hidden, on display are the resulting beam traces. The info entry `comment` contains the actual slowness value for each trace.

It is convenient to change the trace info text to the `comment`–entry (command `trctxt ^comment($x)`).

14 Notes for UNIX Versions of SH

The SH program was originally developed on ATARI ST/TT computers running TOS as operating system and GEM as graphics interface. Since most parts of SH are written machine–independend it was quite easy to export the program to VAX/VMS (after implementation of a VWS and an X–Window interface). The implementation on UNIX, however, shows some peculiarities. This is mainly due to two features of UNIX, namely the occurrence of slashes "/" in filenames and case–sensivity. The slashes are used in SH (as in VMS) to indicate command qualifiers (like the hyphen in UNIX). Therefore in calls to the operating system cannot be used any slashes. This is very annoying if filenames have to be passed. This problem can be solved by the use of the internal variable `$slash` (see section 9) in combination with concatenation expressions (`|$slash|home|$slash|sh|$slash|` instead of `/home/sh/`). This is very clumsy sometimes as you can see in the example. Another possibility is to change the qualifier character to a backslash "\". Then you can use slashes without problems, but all existing command procedures in the library must be changed in this way. I admit that both solutions are not perfect. Similar problems arise concerning the case–sensitivity of UNIX. By default, SH converts every input line to uppercase letters before translating it. This automatic case conversion can be switched off (see section 8.6), but then you have to enter all command verbs and keywords in uppercase letters. This is also not a convenient solution of this problem. I'm still thinking about these things and I hope I can optimize the UNIX interface in the near future.

Please note, that these peculiarities affect only calls to the operating system (command `system`). All other commands behave exactly like in other implementations.