NAME

MASE basic introduction and tutorial session.

SYNOPSIS

This document is intended to help you get started in *MASE.* Many have noted that the *MASE* manual as a whole is usable as a reference document, but that it does little to help one get started. So - here we go! While going through this tutorial, if something is unclear, look it up in the manual (either paper or online). This tutorial is, by no means, a substitute for the reference portions of the *MASE* manual.

Getting Started

To run the editor, we need something to play with - a demo sequence. *MASE* has a standard demo file; to get a copy of it, run ‘‘mase-demo’’; *MASE* will get a copy of this demo sequence file and call it ‘‘mase-demo.pep’’, and place it in your current directory. Let’s get started!

You should have something on your screen similar to picture 1 (your screen may differ, depending on how many lines and columns your terminal can display).

Positioning the Cursor, MULT factor

Try your cursor arrow keys. Hopefully, they will work. If they do not work, it’s probably worth contacting your *MASE* administrator now to see if you can get it rectified.

Hitting the down arrow key should make the cursor go to the next sequence. Now, hit the ‘‘4’’ key. At the bottom of your screen, you should see ‘‘mult = 4’’. Hit the down arrow key again; you will be moved forward four sequences. Try ‘‘1’’, ‘‘2’’, and then the right arrow key. Your cursor should appear twelve positions farther into the sequence. Note how the ‘‘Display Position’’ continuously reflects your current position.

Automatic Windowing

What happens when you move your cursor to the edge of a window? Try it! Keep hitting the right arrow key until your cursor appears on the right most boundary of the left sequence window. Now, hit the right arrow again. You should now be in the right window, on the left-most border. Hit the right arrow repeatedly, this time to position it on the right most border of the right window. One more right arrow now . . . Surprise! The right window will re-adjust itself to allow the current position to be displayed.

What about vertical windowing? Use the down arrow to bring the cursor to the last displayed sequence on this screen; now, hit the down arrow key once more. Again, the screen will adjust itself so that the cursor position is displayed on the screen. Think of the cursor keys as moving you around the *SEQUENCES,* not around the screen; the screen is a complete slave to the cursor and the sequences. Get the picture? By the way - the behavior of *MASE* when window adjustments are required can be modified. See the discussion about the *INTERNAL VARIABLE* named *LOCK WINDOWS.*

The Search function

There are many other commands built into *MASE* which also move the cursor. Try the following set of key strokes:
‘‘/A<CR>’’ (hit the key labeled ‘‘RETURN’’, ‘‘CARRIAGE RETURN’’, or ‘‘ENTER’’ for the symbol ‘‘<CR>’’). Notice where the cursor appears - it found the next occurrence (searching forward from the cursor position) of the pattern ‘‘A’’. we’ve just tried the *MASE* function ‘‘Search’’. More on all this function business later.

The Menu System, Short Help

All of *MASE* works on commands (or functions). A moment ago, we saw that the ‘‘/’’ key started (or called) the ‘‘Search’’ function. The arrow keys called the move functions (‘‘MV-Up’’, ‘‘MV-Down’’, ‘‘MV-Left’’, and ‘‘MV-Right’’). These functions can be called using their long names as well as their associated key strokes.

Functions can be called by their long names from the ‘‘Command Interface’’. This interface is initiated by hitting the colon (‘‘:’’) key. You will see your cursor jump into the command menu that will appear at the bottom of the screen. Notice the reverse video ‘‘M’’ at the left margin. It indicates that this is a menu selection - there is a finite set of possible responses.

This menu contains a list of all the long command names in alphabetical order. The first command is ‘‘@Abort-Command’’; this will be displayed to the right of the prompt arrow (‘‘->’’). Each menu item has a short help text associated with it. This help is invoked by hitting the question mark key (‘‘?’’). Try this now - hit the ‘‘?’’ key. Notice that the help is displayed on the last few lines of the screen, replacing the menu. To walk through this help selection, hit the space bar. This text may fill one or many of these little windows. The reverse video plus sign (‘‘+’’) in the lower right corner means that there is more help remaining to be displayed. When the help has been displayed, you will be returned to the menu.

How do you ‘‘select’’ things from this menu? Try hitting the down arrow key. The function name ‘‘Alphabetize-Lists’’ will be displayed. Try hitting the down arrow key again. The function name ‘‘Apropos’’ will be displayed. Try the ‘‘?’’ help here, just to see what it says, then get back to the menu. So - you can scan the menu list up and down (oops! to go back up in the list, use the up arrow!).

Since *MASE* has over 70 functions, this method has its problems (especially if you frequently needed the function ‘‘Zap-Sequences’’!). The alternate method is to start typing the function’s name. Now, hit the ‘‘M’’ key. The function name ‘‘Map’’ will be displayed. The menu interface uses a little ESP to do what I call incremental completion. ‘‘Map’’ is the first command in the list that starts with ‘‘M’’. Try hitting the ‘‘V’’ key now - the function name ‘‘MV-DOWN’’ will be displayed, the first function name that starts with ‘‘MV’’. Try the down arrow key now. The name ‘‘MV-LEFT’’ will be displayed. You can uses a series of key and cursor key strokes to find an entry in the way most efficient for you. While ‘‘MV-DOWN’’ is displayed, hit the ‘‘<CR>’’ key (remember - hit the key labeled ‘‘RETURN’’, ‘‘CARRIAGE RETURN’’, or ‘‘ENTER’’ for the symbol ‘‘<CR>’’). You will be returned to the sequence, with the cursor moved down one sequence.

Does the repeat factor (a.k.a. ‘‘MULT’’) work with the command interface? Try it! How about trying this: hit the ‘‘5’’ key, then run ‘‘MV-DOWN’’ from the command interface (remember, hit the ‘‘:’’ key to invoke the command interface).

Binding a key to a function

T’would be a pain to move around sequences all day if you had to hit ‘‘:’’, ‘‘M’’, ‘‘V’’, and ‘‘<CR>’’ just to move the cursor down, I suppose. So, there is a mechanism to ‘‘bind’’ a key stroke to a function. If you hit that key while you are in the major editing mode (your cursor is in the sequence), its bound function is called. Want to experiment? O.K. - how about binding the key ‘‘t’’ to the function ‘‘Ins-Gap’’. First off, lets see what ‘‘Ins-Gap’’ does. When you have the name ‘‘Ins-Gap’’ displayed in the command menu, try the ‘‘?’’ help. Good ’nuf? O.K., now run it. See what it does? Now, lets ‘‘bind’’ something. How to start? Run the function ‘‘Bind’’. It will say ‘‘Key to bind’’. Hit the ‘‘t’’ key. It will say ‘‘Function to bind it to’’. Select ‘‘Ins-Gap’’ via the same mechanism as in the command mode. We’re back in the sequences again. Use the cursors to move to a fresh spot in the sequence. Hit the ‘‘t’’ key now. Any surprises? Good! Try ‘‘4’’, then ‘‘t’’. MULT works regardless of the METHOD by which the function was called. The only thing *MASE* is inflexible about is the list of long command names. Key strokes can be bound to any function desired. For convenience, *MASE* has a default set of bindings. To see what they are, run the function ‘‘Show-Bindings’’. A little verbose, eh! You can, at any time, bind a new function to any key; the old binding will just be forgotten. Keep in mind that a given key stroke may be bound to only one function, but that one function may be bound to several (or no) key strokes. By the way - each key stroke *MUST* be bound to some function. Keys that are ‘‘unbound’’ are actually bound to the function ‘‘BELL’’.

Saving your work

When you began editing, the locus names were all displayed in ‘‘regular video’’ - they weren’t highlighted. What about now? See anything unusual?

When a sequence is ‘‘dirty’’ (i.e. when it has unsaved modifications), its name will be displayed in reverse video. Run the function ‘‘Save’’ now. What happened to the locus names? Why? because they’re no longer dirty - all of the modifications have been saved. This is just a little aid to help you see which sequences you’ve touched in the current editing session.

Variables: What are they and how to modify them

What are variables? They are ways to affect the way different functions and editing aspects behave. Lets start with an example (sorry if it gets a little long winded!). Here we go! Go to some fresh place in the sequence. Hit the ‘‘i’’ key (bound, by default, to the function ‘‘Ins-Gap’’). This should insert a gap (actually, a dash) under the cursor. Hit the ‘‘<DELETE>’’ key. The gap should be removed. The ‘‘<DELETE>’’ key is bound to the function ‘‘Del-Back’’. O.K. - the gap was behind the cursor, and ‘‘Del-Back’’ zapped it away. Try hitting ‘‘<DELETE>’’ to delete a regular sequence letter. What! Attempt to delete protected characters ? ? What does that mean? Remember, *MASE* was designed to edit alignments. What is involved in ‘‘editing an alignment’’? Inserting and deleting *GAPS*, but not modifying the sequence elements. With this in mind, the delete family of functions are less willing to delete sequence elements than gaps. Well, what if you really want to zap some of these precious sequence elements? There is a variable called ‘‘Protect’’ which controls this. Lets continue; hit ‘‘<CR>’’ to clear the error message. Lets check out this ‘‘Protect’’ variable. Run the function ‘‘Set-Variable’’. You will be presented with a menu of about 20 variable names. You can move about this list the same way you move and select from the command menu. Also note that ‘‘?’’ will bring up short help messages regarding each individual variable. Bring up ‘‘Protect’’, then hit ‘‘?’’ to see what it’s about. Finish out the help, then press ‘‘<CR>’’ when ‘‘Protect’’ is displayed. It will say ‘‘Set variable (boolean) -> ’’. What is ‘‘boolean’’? It means that this variable can have one of two values, either ‘‘ON’’ or ‘‘OFF’’. As with setting all variables, the current value is offered as the default. Notice that the ‘‘Protect’’ is currently ‘‘ON’’. Notice also that there is a reverse video ‘‘M’’ at the left of the prompt - this is also a menu selection. Pick ‘‘OFF’’ (either with the cursor, or by typing ‘‘OFF’’), and follow with a ‘‘<CR>’’ to select. You will be returned to the major editing mode. Now, try the ‘‘<DELETE>’’ key again. It will now merrily eat up anything offered to it - gaps, sequence elements, the whole lot. Get the picture? Lets see what the current variable values are. Run the function ‘‘Show-Variables’’. See, there is ‘‘Protect’’ (now ‘‘OFF’’). Note that there are three variable types: integers (whole numbers), floating points (real numbers), and booleans (ON or OFF). If you’re curious about what all these variables are, either look them up in the ‘‘Internal Variables’’ section of the *MASE* manual, or browse through the ‘‘Set-Variables’’ menu and use ‘‘?’’ to view the short helps.

Exiting MASE

Lets see what happens when we quit (and *HOW* to quit!). First, hit a few ‘‘i’’s. One or more locus names should be displayed in reverse video, indicating that they have been modified. Run the function ‘‘Quit’’. *MASE* will tell you that there are unsaved modifications (‘‘Some buffers modified’’), and will ask if they should be saved. Lets say ‘‘yes’’. Finally, you will be asked to verify that, yes, you do want to exit *MASE.* You should now be at the main shell prompt.

Starting MASE

Lets start up *MASE* again, and call up our demo sequences. Just run ‘‘mase’’ for now, with no arguments. *MASE* will prompt you for the name of a sequence file to load. Respond with ‘‘mase-demo.pep’’., followed by ‘‘<CR>’’. You will see the locus names and lengths ticking off at the bottom of your screen. Run the ‘‘Quit’’ function to return to the shell prompt. There are no modified buffers, so you will only be asked to confirm your exit. Now, run ‘‘mase mase-demo.pep’’. Again, you will see locus names and lengths ticking off. The purpose? To show you that there are two ways to start *MASE*. Actually, you can specify multiple file names as command tail arguments. *MASE* can edit multiple files simultaneously. On ‘‘Save’’ing your work, only files containing modified sequences will be rewritten. You can, as an alternative to specifying file names as the command tail, use the function ‘‘Load’’ to bring up additional files.

Non-menu Entry and Pattern-Highlight function.

First off, to save time, bind the keystroke ‘‘p’’ to the function ‘‘Pattern-Highlight’’. Refer back to *Binding a key to a function* if you don’t know how to do this.

Good. Let’s see how this function works. Hit ‘‘p’’ to run the function. *MASE* will prompt you first to ask if it should clear the existing highlights. Say ‘‘no’’. You will now be asked for the ‘‘Pattern to highlight’’. Let’s use ‘‘A’’. Hit the ‘‘A’’ key, followed by ‘‘<CR>’’. *MASE* will work for a second or two, then return to the main editing mode. Note that all occurrences of ‘‘A’’ are highlighted. Also note that an asterisk (‘‘∗’’) appeared in the left margin for some sequences. These are the sequences which contained the pattern most recently highlighted. (This might be useful if you are working with long sequences, and you want to see in which sequences pattern was found.)

Hit ‘‘p’’ to call ‘‘Pattern-Highlight’’ again. Notice that ‘‘A’’ is offered as a default. Hit ‘‘C-U’’ (Control and U) to erase this entry, then try ‘‘[ALVI]{2,5}[ˆTFY]’’. Look up the manual pages for the GNU-EMACS regular expression handler. This ‘‘thing’’ handles the wild card (ambiguity) aspects for *MASE*, *GGREP*, *SEQ-VIS*, *SEARCHER*, and other tools. Familiarity with its syntax will make things a lot easier and more productive for you. Onward - what this pattern specifies is that there must be between three and five positions that are ‘‘A’’, ‘‘L’’, ‘‘V’’, or ‘‘I’’, followed by anything except ‘‘T’’, ‘‘F’’, or ‘‘Y’’. Give it a whirl. See where it found hits? Let’s try another. Follow these key strokes; I want you to play with this interface (input method) for a minute. Hit ‘‘p’’. Tell it to clear the old highlights. The last pattern used (‘‘[ALVI]{2,5}[ˆTFY]’’) will be recalled for you. Try the up arrow key. The first pattern, ‘‘A’’ will be recalled. Actually, all of your responses will be kept, so that you may refer to them. Hit the down arrow key to recall the more complex pattern. Now, try the left arrow key. See how the cursor moves? You are in a little one line editor. Try ‘‘C-D’’ (control and D). It should delete *UNDER* the cursor. Try the ‘‘<DELETE>’’ key. It should delete *BEHIND* the cursor. (No, the delete functions down here do not respect ‘‘Protect’’.) Try typing in a letter or two while the cursor is in the middle of the string. See how they are inserted? Try ‘‘C-A’’. How about ‘‘C-E’’? Try ‘‘C-U’’. Oops! Where’d it go? ‘‘C-U’’ will erase the entry. Did we remove that entry from the history list? Hit down arrow, then up arrow. See how it retrieved it again? What happens if you want to cancel a function after you have started it? Right now, hit ‘‘C-C’’. It will say something like ‘‘Function aborted’’.; C-C will abort any function, at any point. Be careful about the side effects of functions aborted while they are in progress - might get a little sticky! You might take a moment to go through the section about the ‘‘Menu Interface’’ in the *Introduction* section of the *MASE* manual.

Editing Sequences

What if we want to edit more than just ‘‘the alignment’’? What if we want to insert sequence elements? Remember the ‘‘Bind’’ function? There is a function called ‘‘Auto-Insert’’. What ever key was struck to invoke this function will be inserted under the cursor. (This function isn’t useful unless keys are bound to it - calling it from the command menu will insert a colon (‘‘:’’). Bind the key ‘‘A’’ to the function ‘‘Auto-Insert’’. Bind the key ‘‘t’’ to the function ‘‘Auto-Insert’’. Now, while in the major editing mode, hit the ‘‘t’’ key. See how the ‘‘t’’ was inserted into the sequence? Hit ‘‘A’’. Notice that case is preserved. Lets cheat a little. Run the function ‘‘Take’’; the menu has been pre-loaded with some file names for you.

(Might be a good idea to look up the documentation on ‘‘Take’’ in the Intrinsic Functions section of the *MASE* manual now.) Use the arrow keys to pick the entry ‘‘=/keys.protein’’, and hit ‘‘<CR>’’. This will load a set of key mappings and bindings to set up the keyboard for editing protein sequences. Hit ‘‘a’’. Hit ‘‘L’’. Notice that everything goes in as upper case.

Remember the variable ‘‘Protect’’. If you are entering sequences, you will most likely want to turn ‘‘OFF’’ ‘‘Protect’’ so you can delete your mistakes.

The ‘‘Output-Aligned’’ function

(I assume that, at this point, you are running *MASE* on the demo file.) The output function is a complex beast. What if you just want a simple printout; forget all the bells and whistles? And, how do you get at some of the fancy aspects? How do you get it configured for your printer?

Lets start with a simple, no frills case - no fancy stuff, and leave all the parameters at default. Run ‘‘Output-Aligned’’. When it asks ‘‘File containing mappings’’, respond with ‘‘NONE’’. When it asks ‘‘Map by patterns’’, hit ‘‘<CR>’’. Now, it will ask for the name of the output file. If the first character of the file name you type is an exclamation point (‘‘!’’), then the rest of the string will be a command that will receive the formatted sequences. It will get them as STDIN - standard input. If you want the output to go, for example, directly to the printer, use ‘‘!lpr’’ (use any arguments to ‘‘lpr’’ that you want). This is nice - it doesn’t generate a temporary file, and will make maintaining your directory a little easier. As *MASE* generates the output, it will tick off each page as it is finished.

‘‘Output-Aligned’’: printer configuration

Before we start more complex examples of ‘‘Output-Aligned’’, we have to get a configuration file for your printer. If you have a DEC printer (LA-100 or such), you are in luck - I’ve made a configuration for the LA-100 that will probably work for you. To build a configuration, you will need to know the character sequences (escape sequences, whatever you want to call them) to make your printer display two distinct text styles. Type styles you may find useful might be regular, bold, underlined, super-script, sub-script, italic, or other things like that. Have these two character sequences in hand? O.K. - get out of *MASE*. Use whatever editor you like (vi, emacs, whatever) to create a configuration file. Give it whatever name you like. We’ll be creating three lines. Each line will have two parts. The first part is the one character ‘‘key’’, or index. The second part of each line, from the second character to the end of the line, is the character sequence that is to be sent to the printer. One line will have a space as the first character (the ‘‘key’’ will be a space). The sequence associated with this key will be used to ‘‘reset’’ the printer before printing numbers, page labels, locus names, and such. Another line will have a zero (‘‘o’’) as the key. The sequence associated with ‘‘0’’ will be sent to the printer before non-highlighted positions are sent. The last line will have a one (‘‘1’’) as the key. The character string associated with ‘‘1’’ will be sent to the printer before the highlighted positions are sent. Now - how do you specify these ‘‘character sequences’’? For starters, flip to the discussion of ‘‘string conversion’’ in the Introduction section of the *MASE* manual (near the end of this section). *Read this section!* Suppose we are using the ‘‘SGR’’ codees. (These are from an ‘‘ANSI’’ definition, and are used for some printers, including DEC). This code specifies the sequence ‘‘^[[4m’’ (escape, left square bracket, the digit four, and an ‘‘m’’) to start underlining, and ‘‘^[[m’’ to stop underlining. The way the configuration file would be set up, then, would be:

```
 \e[m
0\e[m
1\e[4m
```

Note that the first line begins with a space. Get the picture? When you think you’ve got this file configured, you can get back into *MASE*.

‘‘Output-Aligned’’: a more complex example

First off, we have to highlight something. Use ‘‘Pattern-Highlight’’ to highlight the pattern ‘‘[ALVI]{3,}’’. Run ‘‘Output-Aligned’’. It will ask for the name of the file containing the mappings. If you want the DEC sequences, use the cursor to pick ‘‘=/colors.la100.patterns’’. If you created your own configuration file in the above step, type in that name. Select ‘‘Map By Patterns’’. Pick whatever you want for the output file name. If you use ‘‘!lpr’’ or something like it, make sure that the output is going to a printer appropriate to the mappings you’ve used.

A last little note on the output.

The function ‘‘Pattern-Highlight’’ will accept a ‘‘MULT’’ factor. Rather than being used as a repeat factor, it identifies patterns as being ‘‘different’’ for the ‘‘Output-Aligned’’ function. So, if you hit ‘‘2’’, then run ‘‘Pattern-Highlight’’, the sequence areas that match will be called ‘‘pattern 2’’. The character sequence associated with the key ‘‘2’’ in the printer configuration file will be sent to the printer before the match is printed (rather than the sequence associated with ‘‘1’’). This feature is most useful if you have access to a color printer. ‘‘MULT’’ factors from ‘‘1’’ through ‘‘9’’ are currently available toward this end.