# MTNEW
# Multichannel signal display

**Typographical conventions in this document:**

matlab functions and variables, and mtnew command names are shown in the bold, fixed-width font, e.g **`data, eval, display_settings`**

Prompts from the program to the user are shown in the red, fixed-width font, e.g. **`M>, enter blank-separated list`**

Input from the user to the program is shown in the green, fixed-width font, e.g **`v, ?, JAW`**

Names of signals are shown in the magenta fixed-width font, e.g **`AUDIO`**

## Getting started

The signals that are displayed in mtnew are stored in MAT files. A complete recording session (experiment) is divided into trials, which are identified by a number from 1 to n. Each MAT file only contains data for one trial. However, there may be more than one MAT file per trial. In general, different categories of signal (e.g EMMA vs. sonagram vs. video) and signals with different sample rates (e.g EMMA vs. audio) are stored in separate files.

For example, in an imaginary experiment in which EMMA and audio was recorded the data might be stored in MAT files called myexp_emma_001.mat (EMMA data trial 1), myexp_audio_001.mat (audio data trial 1), myexp_emma_122.mat (EMMA data trial 122) and so on.

**`mtnew`** is a matlab function that must be called with a number of input arguments. These are outlined in this section. This section also briefly describes the basic way of interacting with the program.

The main part of this document then describes each command available in the program.

Separate chapters give detailed information on the structure of the mat files used, on advanced ways of interacting with the program, and on programming extensions to the program.

The program can in principle be used without any knowledge of matlab programming. However, it can be put to much more effective use with such knowledge. The program is built up on a set of matlab functions that are designed to make it easy for users to add their own extensions to the program.

### Starting the program

**`mtnew`** is called as follows:

**`mtnew(cutfile,recpath,reftrial,signalspec,cmdfile,session_number)`**

The first four input arguments are compulsory, the last two are optional.

| | |
|---|---|
| **`cutfile`**: | MAT segmentation file containing (at least) variables **`data`** and **`label`** (further details are given in the chapter on MAT file structure) |
| **`recpath`**: | Common part of signal filename |

**`reftrial`**: Number (as string; e.g. '001') of reference (or typical trial). The program examines the files related to this trial to get basic information on the signals to be displayed (a further use is to tell the program how many digits are used to code trial number in the file names).

**`signalspec`**: This is the most complicated specification that needs to be made. **`signalspec`** is a string matrix, in which each line provides information on one signal. The information is divided into up to four fields, separated by a period:

**"unique_filename"**

i.e the unique part of the the mat file name. In the example given above, 'recpath' would be set to 'myexp_' (probably prefixed by the path to the files), and "unique_filename" for the audio data would be 'audio_', and 'emma_' for all the EMMA signals to be displayed. In other words the "unique_filename" is the part of the filename between the part defined in 'recpath' and the trial number.

"**external_signalname**"

Name of signal in the mat file. Each mat file contains a variable called **`data`**, containing the actual signal data. Multichannel data is arranged so that each column of **`data`** constitutes one signal. The name of the signal in each column is defined in a string matrix variable called **`descriptor`**, which has as many lines as there are columns in **`data`** (there is an exception to this rule). Thus you need to know these names in order to identify the required signals to the program. If necessary, you will have to examine **`descriptor`** in one of the data files before starting the program.

"**internal_signalname**"

Name by which the signal is referred to during the **`mtnew`** session. This is optional; it defaults to "external_signalname". However, it may often be convenient to define a new name for use within the progam: The names in "external_signalname" may be cumbersome to use (they are sometimes generated automatically). e.g suppose an external signalname is 'TONGUE_TIP_X'. You could assign this the shorter internal name 'tipx'.

Sometimes it may even be essential to assign a new internal name. For example, you may have separate mat files with two different versions of an f0 track, but with an identical name in the descriptor variable in each file, e.g both use the signalname 'F0'. In order to look at these signals together you will need to give them different internal names.

The syntax is thus:

unique_filename.external_signalname[.internal_signalname]

Example:

To load audio and tongue-tip data, giving a new internal name to the EMMA data:

str2mat('audio_.AUDIO','emma_.TONGUE_TIP_X.tipx','emma_.TONGUE_TIP_Y.tipy')

(**`str2mat`** is a matlab function that gets a list of strings into the arrangement required by matlab)

There is a fourth optional specification called "mem_mode". This is mainly intended for use with video data, and allows such data (which is typically stored in an 8 bit format in the mat files) to be also stored internally as 8bit, rather than being converted to double when loaded, which is the default behaviour of the program. This is discussed in detail in the chapter on MAT file structures.

**`cmdfile`**: Optional: Command file name. This is a text file of commands to the program, which are processed when the program is started. This is extremely useful, as it avoids having to laboriously type in large numbers of display options by hand every time the program is started. Basically, a command file can simply consist of a list of commands (one to a line) exactly as the user would have entered them at the keyboard. However, command files have a number of additional features that are discussed in the section on advanced interaction with the program. In order to provide the framework for a commandfile, Matlab's **`diary`** function can be used to record the commands entered at the keyboard (a small amount of editing of the diary file will normally be necessary before it can be used as a command file).

**`session_number`**: Optional. It is perfectly possible to run multiple matlab sessions simultaneously, with mtnew running in each matlab session. In order to make it possible to identify which figures belong to which session, an optional (arbitrary) session number can be specified. This is shown in the title bar of each figure (and also in the main level prompt, in the command window)

## Overview of main display types

**`mtnew`** uses separate matlab figures (windows) for the different kinds of displays.

Each type of display (each figure) has a name; a brief description (with their names) of the currently available displays is given in the following list. The display name is to be found in the title bar of the figure (in the form "mt_displayname_sessionnumber").

| **Name** | |
|---|---|
| **f(t)** | Waveform display |
| **xy** | Trajectory display. Actually more versatile than just a simple 2D x/y display. Allows flexible use of 3 spatial axes plus a colour axis. Also used for animation of movements. The figure can be subdivided into any number of individual trajectory |

|  |  |
|---|---|
|  | displays, each with its own specifications (e.g simultaneous sagittal and coronal views of 3D data). |
| **sona** | Sonagram display. Not restricted to the classic sonagram representation of spectral data from the speech signal. Can be used for any data that is formally equivalent to a time-series of spectral slices (e.g a special kind of EPG display is possible). As with the xy display, the figure can be subdivided at will, e.g to show both wide- and narrow-band sonagrams. Note: Sonagrams are not generated by the program. They must be computed and stored as MAT files beforehand (see MTSONA2FILE). This is a deliberate part of the philosophy of clearly separating signal acquisition or generation from signal display, and ensures the above-mentioned goal of making this display type available for any signal that is formally equivalent to a sonagram. |
| **video** | Video display. More formally, it can be used for showing films of any data that can be regarded as a time-series of images. Again subdivision of the figure is possible. |
| **organization** | The default setting of this figure is to show a rough oscillogram of the audio data (if available) of the current cut. It can also be set up to show various other kinds of 'background' information. This is detailed elsewhere. Normally, this is the display with which the user will be least concerned. |

For more details, and examples of typical displays see ???

When mtnew is started only the organization and f(t) figures are created. The other display types are created by the user if and when they are required.

## Program Commands: Introduction

Program commands are selected by the user by typing the command letter assigned to it (user input is made via the normal matlab command window, except when in cursor mode (see below)). The command must be terminated by typing `<CR>`. Commands will normally be referred to by the full command name, since these are fixed, rather than by the single letter used for activating them, since this assignment of letters to commands can in principle be chosen freely by the user (customization of these assignments of keys to commands is discussed elsewhere).

When the program starts, the prompt in the command window is `M>`. This indicates that the user is at the **`Main`** command level. The prompt changes whenever a command is given that takes one to a new command level (For example, the prompt changes to `md>` when the **`display_settings`** command is chosen at the **`Main`** command level, and changes to `mdx>` when the **`setxy`** command is chosen at the **`display_settings`** level. Thus the length of the prompt indicates how far away one is from the main command level. Usually it is possible to back up a level in the command structure by simply pressing `<CR>`.)

The command structure of the program can be viewed in the menu bar of the organization figure. The menu items shown there are the **`Main`** level commands. Most of these commands expand to show further levels when they are clicked on. Note: These drop-down menus in the organization figure are for information only. Clicking on any of the menu items does not actually execute the command! However, clicking on any menu item that is a terminal item in the menu tree (i.e which does not expand to a further submenu) will call up the help text for that command. The underlined letter in the command name is the letter that must be typed at the matlab prompt to execute the command. If the command letter is not in the command name it is shown to the right of the command name.

A more direct way of getting help when interacting with the program is as follows:

Whenever input is required from the user, a list of the choices available can be obtained as a pop-up menu by typing `?`, and the choice can be selected (and executed) by double-clicking on it. This is rather a slow way of entering commands themselves; after some practice with the program this technique will probably not be needed much. However, as a result of some commands the user is prompted, for example, to enter a list of signals for display etc. The `?` technique works here too; this will give a pop-up list of the signals that can be selected.

Further notes on entering lists at the keyboard:

(This is also particularly relevant when using a commandfile, i.e a situation where pop-up lists cannot be used.)

When entering a list at the keyboard (the user is prompted for a `blank-separated list`), a useful feature is that partial specifications can be used: If, for example, `AUDIO` is the only signal whose name starts with the letter “A” then it is sufficient to simply type `A`. This mechanism also allows multiple signals to be selected with a single specification: If there are signals called `JAWX` and `JAWY` then typing `JAW` (or possibly even just `J`) will select both of them. (Note: If the full name of one signal forms the first part of the name of another signal only the first signal will be selected; i.e multiple selections are only made if an exact match is not found.)

It is also possible to choose items in the list by giving a vector of indices in the list (using any standard matlab syntax). Enclose the vector in ‘[ ... ]’.

e.g `[1 3 5], [3:6], [1:3:7].`

Use can also be made here of a variable named `ls` that indicates the last item in the list: e.g `[5:ls]` chooses the fifth item up to the last item.

## Common commands

There is a group of commands that is referred to as “<Common>” in the menu bar of the organization figure. Commands in this group can be used at any command level in the program. They are never shown explicitly in the list of commands available. In fact the `?` technique is an example of such a Common command. Strictly speaking it should be referred to as the **`what`** command. The letter `?` is simply the preset command letter for activating this command. The other main help commands are also Common commands: **`global_help`** (default letter `H`) uses the browser to open this PDF document; **`local_help`** (default letter `h`) displays in the Help window the detailed descriptions of all commands available at the current location in the command hierarchy. A complete list of the common commands is given below.

## Changing Graphics properties

The program does not provide ready-made commands for manipulating properties of the graphics objects like line styles and widths, marker symbols, and axis grids.

However, whenever the program is waiting for user input at the main level (prompt `M>`) then it is possible to call up matlab's graphics property editor by clicking on any graphics object. This offers a great deal of flexibility and avoids having to program many little details into **`mtnew`** itself.

It can be particularly useful when preparing figures for hardcopy.

However, this feature should be used with care. It is best to confine any edits to 'simple' properties like the ones mentioned above. Some properties of the graphics objects should not be changed under any circumstances (especially **`userdata`** and **`tag`**) as they are essential to correct operation of the program.

## Detailed Description of all Commands

Under construction!!

Should eventually include details of all commands given in the list of commands at the end of this document. Currently only complete for cursor-level commands.

## Quickstart

MTNEW: Version 23.08.02

Assuming default command key assignments, at any program prompt type:

| | |
|---|---|
| ? | to get a list of currently available choices |
| h | to get a detailed description of currently available commands |
| H | to open the full on-line documentation |

Use the menu bar in the mt_organization figure to view the command hierarchy.

Click on any terminal member in the hierarchy to obtain a description of that command

Many cursor movement commands are assigned by default to numeric keys.

Engage the numlock key if you want to use the numeric keypad for cursor movement (but you will need to disengage it on Linux systems to move and resize figures).

## Cursor

### Introduction

Unlike all other commands, commands at this level are carried out immediately when the command key is pressed. When the cursor level is activated two cursors become visible in the f(t) figure. If a sonagram figure has been activated, linked cursors will be visible here, too. If an xy display has been activated, the time instants corresponding to the two cursors are marked by contours (e.g of the tongue) linking up the time-points on the trajectories. In all these figures, one of the cursors is green and one is red. The green one is referred to as the active cursor. Many commands operate specifically on the active cursor.

In order for the cursor commands to work, the mouse pointer must be in the f(t) figure and the figure must be active (i.e title bar is highlighted). In fact, it is possible to designate other figures as the target figure for cursor commands (this is done with the main>display_settings>foreground command). This is most likely to be useful for the sonagram figure, so that the mouse can be moved over the sonagram to position the cursors. It is not recommended to set a non-time-based figure like the xy figure or a video figure to be the foreground figure.

Cursor commands are mainly used for moving the cursors around (surprise!). In addition to some miscellaneous commands there are two important submenus, one for sound, and one for defining segment boundaries and labelling the signal (= marker commands).

A note on the cursor movement commands: It should be obvious, but don't forget that the right cursor can never be moved to the left of the left cursor, and vice-versa. The default key assignments have been based on use of the numeric keypad with the right hand, and movement of the mouse with the left hand, with left and right buttons positioning left and right cursors respectively.

### leftcursor2pointer

Moves the left cursor to the current position of the mouse pointer.

### rightcursor2pointer

Moves the right cursor to the current position of the mouse pointer.

### slowleft

Moves the active cursor to the left in small steps. One step is defined as 0.001 times the current length in seconds of the f(t) display.

### fastleft

Moves the active cursor to the left in large steps. One step is defined as 0.01 times the current length in seconds of the f(t) display, i.e 10 times the size of the slowleft command.

### slowright

Moves the active cursor to the right in small steps. One step is defined as 0.001 times the current length in seconds of the f(t) display.

### fastright

Moves the active cursor to the right in large steps. One step is defined as 0.01 times the current length in seconds of the f(t) display, i.e 10 times the size of the slowright command.

## jumpleft

Moves both cursors to the left by the time corresponding to the current distance between the cursors. Thus the right cursor moves onto the previous position of the left cursor, and the distance between the cursors does not change.

## jumpright

Moves both cursors to the right by the time corresponding to the current distance between the cursors. Thus the left cursor moves onto the previous position of the right cursor, and the distance between the cursors does not change.

## swap

Swaps the active cursor, i.e the red cursor turns green and the green one red.

## jumpmstart

Move active cursor to current start marker.

This command will only have any effect if marker commands have been used to define segment boundaries. See discussion of marker commands for further background. The command will also have no effect if  e.g the active cursor is the left cursor and the current start marker is to the right of the right cursor (and vice-versa).

## jumpmend

Move active cursor to current end marker.

This command will only have any effect if marker commands have been used to define segment boundaries. See discussion of marker commands for further background. The command will also have no effect if  e.g the active cursor is the left cursor and the current end marker is to the right of the right cursor (and vice-versa).

## previouszx

Move active cursor to previous zero crossing in the audio signal. (Note: In fact any signal can be used as the audio channel. See main>i_o>audiochannel.)

## nextzx

Move active cursor to next zero crossing in the audio signal. (Note: In fact any signal can be used as the audio channel. See main>i_o>audiochannel.). Currently, only the positive zero-crossing can be used.

### previoussubcut

Moves the left and right cursors onto the start and end boundaries, respectively, of the last subcut whose start boundary is to the left of the current left-cursor position. If no subcuts are completely within the current f(t) display, the cursors are not moved.

Subcuts (and how they differ from markers) are explained elsewhere.

See main>display_settings>settime>sub_cut_control for information on activating subcut display.

### nextsubcut

Moves the left and right cursors onto the start and end boundaries, respectively, of the first subcut whose start boundary is to the right of the current left-cursor position. If no subcuts are completely within the current f(t) display, the cursors are not moved. Also, the command may not work quite as expected if the next subcut is not completely on the screen, but a later one is. Here, too, the cursors will not be moved.

Subcuts (and how they differ from markers) are explained elsewhere.

See main>display_settings>settime>sub_cut_control for information on activating subcut display.

### xyview

Changes the view of displays in the xy figure. For 2D displays of 3D data (e.g sagittal, coronal or transversal views) it is often convenient to design the display as 3D and then switch between the possible views, rather than set up completely different displays for the different viewing planes.

After giving this command the user is prompted for the desired view. Give a 2-letter string (e.g ‘yx’, ‘xz’) to specify the real-world axes to assign to the display x and y axis respectively (it is also possible to revert from a 2D display  back to the default 3D display). Axis direction can be reversed by prefixing the axis letter with a ‘-‘, e.g ‘-xy’.

If there are multiple displays in the xy figure, choose the axes system to be modified by clicking in the coordinate system in the xy figure (it will then be necessary to reactivate the f(t) figure by clicking in its title bar). If this is inconvenient (e.g if you want to set the view in a startup command file) consider using the underlying function MT_SXYV directly (consult the function’s help for further information).

### imageatcursor

Display video frame corresponding to the time-instant of the active cursor (only available if a video figure is operational)

### toggleautoimageupdate

Determines whether video frame is automatically updated whenever the active cursor is moved, i.e without having to explicitly use the imageatcursor command. Initially, this feature is not in operation (it may make cursor movement rather sluggish, so it may not always be desirable).

(Only available if a video figure is operational.)

## return

Exit from cursor mode and return to the main command level.

## sound

### Introduction

Activates the sound submenu (see cursor>sound).

The signal currently defined as the audio channel will be output via the soundcard. Any signal can be assigned to the audio channel (see main>i_o>audiochannel), but the program tries to make an intelligent? initial choice (i.e it looks for a signal called 'audio' or 'Audio'). Assigning non-audio signals like laryngograph or EMG can actually be quite useful.

It is not possible to interrupt the sound output once it has started. However, if any mouse button is pressed, the sound function itself is terminated and new commands can be entered by  the user. In particular, the time instant in the sound signal at which the mouse button was pressed is noted by the program, and it is possible to set the time display in the f(t) figure with this time point at the centre of the screen by using the main>timeshift>bookmark command (see that command for a detailed example).

### cursor

Plays the segment delimited by the current left and right cursor positions

### screen

Plays the currently visible f(t) screen

### cut

Plays the whole of the current cut

### trial

Plays the whole of the current trial (in simple cases where no segmentation has been carried out, this will be the same as the cut command

### current_marker

Plays the segment delimited by the start and end boundaries of the current marker type. If either the start or end boundary has not been set the active cursor is used instead, in which case it must be in an appropriate position, i.e if only the start boundary has been set then the active cursor must be to the right of this boundary (and vice-versa if only the end boundary has been set).

For more background on markers see the marker submenu.

**numbered_marker**

Plays the segment delimited by the start and end boundaries of the desired marker type. After this command has been given, the user must type a single digit to choose the marker type to be played (there is no prompt for this choice, and the key pressed is not echoed). Obviously, if more than 10 marker types are in operation only marker types 1 to 9 can be played with this command. If either the start or end boundary has not been set the active cursor is used instead, in which case it must be in an appropriate position, i.e if only the start boundary has been set then the active cursor must be to the right of this boundary (and vice-versa if only the end boundary has been set).

For more background on markers see the marker submenu.

**left_screen**

Plays from the start of the f(t) screen up to the left cursor.

**right_screen**

Plays from right cursor up to the end of the f(t) screen.

**left_cut**

Plays from the start of the current cut up to the left cursor.

**right_cut**

Plays from right cursor up to the end of the current cut.

**xleft_screen**

Plays from the start of the f(t) screen up to the right cursor.

("x" stands for "extended", i.e unlike the corresponding commands without "x" in the command name this command (and its counterparts like xleft_cut) plays up to the cursor furthest from the start position.)

**xright_screen**

Plays from left cursor up to the end of the f(t) screen.

#### xleft_cut

Plays from the start of the current cut up to the right cursor.

#### xright_cut

Plays from left cursor up to the end of the current cut.

## marker

### Introduction

Activates the marker submenu (see cursor>marker).

This menu allows markers to be set, moved and deleted, and the resulting segments to be labelled.

See the separate section on cuts, subcuts and markers for more background to marker files.

Currently, there is no command to activate the marker functions. It is necessary to call function MT_IMARK directly. See the help for this function for more details. Basically, it is necessary to specify a marker file, the number of marker types to be used, and whether the program is to work in append, edit, or read-only mode.

Marker type is simply an integer from 1 to n. It is not possible to define more than one segment of the same marker type within the same cut. Workarounds for this restriction are discussed elsewhere.

### set_start

Sets a start marker at the position of the active cursor, using the current marker type. The marker is shown in the f(t) figure (and the sona figure if present) as a solid line going from the bottom of the figure up to a height that depends on cut type. The solid line has a right-pointing triangle as line-marker. The position of the start marker is also shown in the organization figure with green right-pointing triangles, with position related to the oscillogram of the current cut. This display can be useful for keeping track of what markers have been set in cases where the f(t) figure does not display all the current cut.

If the start marker has already been set, it is moved to the active cursor position. In other words, it is not necessary to clear the marker to set it again. This means that it is necessary to be a little bit careful not to reposition a marker inadvertently. In future, a mechanism for locking markers to guard against this may be introduced.

### set_end

Sets an end marker at the position of the active cursor, using the current marker type. The marker is shown in the f(t) figure (and the sona figure if present) as a dashed line going from the bottom of the figure up to a height that depends on cut type. The dashed line has a left-pointing triangle as line-marker. The position of the end marker is also shown in the organization figure with red left-pointing triangles, with position related to the oscillogram of the current cut. This display can be useful for keeping track of what markers have been set in cases where the f(t) figure does not display all the current cut.

No end marker will be set if the active cursor is to the left of the corresponding start marker (if already set).

**clear_start**

Clears the start marker of the current marker type.

**clear_end**

Clears the end marker of the current marker type.

**decrease_type**

Decreases the current marker type by one (if the current maker type is already 1, the program issues a warning but does nothing). The current marker type is indicated by a horizontal dashed line across the f(t) figure, and a corresponding number on the y-axis on the right side of the figure. It is also shown in a similar way across the oscillogram of the current cut in the organization figure,

**increase_type**

Increases the current marker type by one (if the current maker type is already at the maximum value, the program issues a warning but does nothing). The current marker type is indicated by a horizontal dashed line across the f(t) figure, and a corresponding number on the y-axis on the right side of the figure. It is also shown in a similar way across the oscillogram of the current cut in the organization figure,

**set_label**

The user is prompted to enter a label. This is displayed near the start marker of the current marker type. It is actually quite possible to define a label even if the corresponding start marker has not yet been set. It is displayed at a time location of zero until a start marker is set (and thus may well not be visible on the screen). (Note that labels are not displayed in the organization figure.)

Currently no special fonts (e.g phonetic) can be used in the label.

**set_type**

Set marker type to a specific type. After giving this command the user must enter a single digit (1 to 9) specifying the marker type to use (thus restricted to cases where no more than this range of markers is in operation). This single digit specification is not explicitly prompted for, and is not echoed. If it is necessary to specify marker types outside this range, then the underlying function mt_smark must be used directly (e.g in a macro). See the function help for more details.

# Advanced techniques for interacting with the program

User input is set up so as to make it easy to assemble frequently needed collections of commands to the program in command files, and to assign a series of elementary commands to 'macro' commands. Thus, if the program does not have a command that you would find useful, consider how existing commands could be combined to provide what you need.

One of the main points to understand here is the use of some special characters. This applies regardless of whether input comes from the keyboard or from a commandfile.

| | |
|---|---|
| `%` | When first character in line, treat this line as a comment (mainly useful for adding comments in command files) |
| `#` | Separates commands on a single line of input. Useful for making command files more legible, but also for defining macro commands |
| `*` | Indicates default input when used as single character between two `#` |
| `@` | If this is the first character on a line of input, then the line is passed to the matlab **`eval`** function before proceeding. The result of the **`eval`** function should be a string of program commands. |

This can be an elegant way of e.g repeating a command (or group of commands) a large number of times.

Examples:

1.

The movie command prompts for the delay factor for the animation

use `@repmat('4#',[1 20])` to have the animation played 20 times with delay factor 4

2.

Scroll through the time display in time steps, waiting for a keypress after each shift

```
@repmat('t#s#q#v#pause#',[1 10])
```

Even this may be awkward to type repeatedly. So if you are lazy or want more flexibility you can prepare this in a small function e.g

```
function ss=r(irep)
myrep=10;
if nargin myrep=irep; end;
ss=repmat('t#s#q#v#pause#',[1 myrep]);
```

Then you just need to type "@r" to get 10 steps through the signal, or "@r(22)" to get 22 steps.

`‘’` If first and last character of a line of input are single quotes then the line is passed directly to the program without being parsed for any of the above special characters (the line is also stripped of the quotes).

An important case where this is necessary is when defining a macro command from the keyboard. Since macro commands normally contain `#` to separate individual commands, the normal parsing needs to be disabled when specifying the macro commands.

## Defining macro commands

There is currently no particular function for defining macro commands.

The definition is made by assigning the command string to a field of variable **`M`**, that is accessible whenever the **`eval_command`** command (its default command letter is `v`) is used.

e.g

```
M.n='n#*#s#X#r'
```

defines a macro called ‘n’ that moves to the next cut and does an XY display of the whole cut

The best way of defining frequently used macros is to write a small script contain a collection of lines of the form

```
M.fieldname1=.......
M.fieldname2=.......
```

etc.

and run this script automatically using the **`eval_command`** command in a startup commandfile.

e.g

```
v
mymacrodef
```

(or `v#mymacrodef`)

If you want to define macros ‘on the fly’ at the keyboard there is a slight complication if the macro contains `#`.

After using the **`eval_command`** command, the `M.fieldname=`.... string must be enclosed in quotes.

i.e you will need to type something like:

```
v
‘M.n='n#*#s#X#r'‘
```

Macro commands also work at the cursor level.

Currently the `@` (and `%`) special characters cannot be used as the first characters of macros.

However, it is quite possible to define a macro with repetitions of a group of commands using a similar construction to the one used in the `@` example above. e.g

```
M.r=repmat('s#s#',[1 20])
```

When used at the cursor level this will give output sound of the current segment 20 times.

This is because the macro definition is done by passing the complete expression to the MATLAB eval function. So the definition can include any MATLAB expression that will evaluate to a string.

**Further points**

1. Reassigning the special characters

To assign the special characters given above to different characters either edit PHILCOM.M (for permanent changes) or write a function to modify the appropriate global variables defined in PHILCOM.M (for temporary changes).

2. Calling command files

The most commonly used commandfile is one specified when calling MTNEW (referred to as a start-up commandfile). However commandfiles with collections of commands can be used at any time in the program:

specify `philcom('commandfile')` from either the **`eval_command`** or **`keyboard`** command.

# Command List
## (with default key assignments)

```
[main]
n next
i i_o
s show
c cursor
t timeshift
d display_settings

[common]
h help
? what
k keyboard
v eval_command
+ eval_variable

[main>i_o]
r return
a audiochannel
p plot
q quit

[main>cursor]
1              slowleft
4              fastleft
7              jumpleft
2              slowright
5              fastright
8              jumpright
0              swap
<left_button>  leftcursor2pointer
<right_button> rightcursor2pointer
[              jumpmstart
]              jumpmend
z              nextzx
Z              previouszx
c              nextsubcut
C              previoussubcut
w              xyview
i              imageatcursor
I              toggleautoimageupdate
r              return
```

```
s               sound
m               marker

[main>timeshift]
q return
0 t0
d duration
s skipforwards
S skipbackwards
l start2left
L end2left
r start2right
R end2right
b start2start
B end2end
[ start2mstart
] start2mend
{ end2mstart
} end2mend
m bookmark
z zoomin2cursor
Z zoomout2cut
c centre2cutpos
C startend2cutpos

[main>show]
r return
t timedisplay
x xycursor
X xyscreen
s sonacursor
S sonascreen
m moviecursor
M moviescreen
f videocursor
F videoscreen

[main>display_settings]
r return
f foreground
x setxy
t settime
o setorganization

[main>display_settings>settime]
r return
```

```
i initialize
c choose_signals
p pair_axes
P cancel_pairing
s sub_cut_control
g get_settings
u up
d down
e extremes
x clipping

[main>display_settings>setxy]
r return
i initialize
f figure_settings
a axes_settings
x axis_settings
g get_settings

[main>display_settings>setorganization]
r return
t maxmin_trial
c maxmin_cut

[main>cursor>marker]
s set_start
e set_end
S clear_start
E clear_end
d decrease_type
u increase_type
t set_type
l set_label

[main>cursor>sound]
s cursor
S screen
c cut
t trial
3 current_marker
6 numbered_marker
l left_screen
r right_screen
L left_cut
R right_cut
[ xleft_screen
] xright_screen
{ xleft_cut
} xright_cut
```

Supplement

Display_settings
(preliminary version, 27.6.07)

## Introduction

The commands in this section are mainly used to set up the display of the time-wave or xy display. A few settings are also available for the organization figure. Currently, there are no interactive commands for adjusting the display in sonagram or video figures, but a few utility functions are available for common tasks.

## setxy

The commands in this section are used to setup and adjust the display of signals in the xy figure. The term ‘xy’ is actually a misnomer, since the displays can be three-dimensional if desired, and can also use colour as a fourth dimension.
The settings can become quite complicated, since there are several possibilities for specifying the kind of data associated with each axis (x, y, z, colour), so it is usually best to include all the xy settings in a command file.

### initialize

This command creates the xy figure (if one is already present it will be cleared).
The user is then prompted to specify the names of the axes to be created (the default is simply ‘xy’). Note that the figure can contain any number of axes, for example to show both a sagittal and axial view of 3D EMA data, in which case a possible specification would be ‘sagittal axial’. These names must be used in some display commands to identify the axis to which the command is to be applied. If more than one axis is specified then the user must specify how the axes are to be arranged within the figure, using a two-element vector as in matlab’s subplot function, i.e [1 2] indicates two axes side-by-side, [2 1] indicates axes arranged vertically.
Following initialization it will be necessary to carry out various axes and axis settings as explained below.

### axes_settings
These are settings that apply to a complete axes system, e.g the sagittal display. The axis_settings command, on the other hand, involve settings that apply to only one axis of one display (e.g x, y, z).
After choosing the axes_settings command it is necessary to choose the display property that is to be modified. These are explained here.

#### n_trajectories
After intialization, the first property that should be set is ‘n_trajectories’. This specifies, for example, how many EMA sensors will be used in the display.

#### trajectory_names
Normally, specification of ‘n_trajectories’ should immediately be followed by

‘trajectory_names’. This would then be the list of the names of the EMA sensors (the precise way in which the names are most conveniently specified may depend on how the axis settings are done (see below). In practice, if the data consists of e.g signals ‘tb_x’ and ‘tb_y’, then the sensor name would normally be specified not as ‘tb’, but as ‘tb_’.

Unlike the above two properties (which must be specified) a specification for the remaining properties is not essential, and will often not be required.

contour_order
This property is useful for specifying whether and how lines joining up the sensors are to be shown. Default if to join up all sensors in the order in which they were specified for ‘trajectory_names’, but it will often be desired, for example, to avoid having the contour of the tongue joined up to the lips. Basically, the specification consists of a list of sensor names, but inserting the keyword ‘<line_break>’ at locations where the contour is to be interrupted (note that it is perfectly possible to specify a sensor more than once).

hold_mode
When set to ‘on’, trajectories already displayed on the screen are not deleted when new data are displayed.

surface_mode
This allows all corresponding time-points in each trajectory to be joined up by setting to ‘row’. This usually only useful when there are only a few time-points in the currently displayed trajectory.

axis_settings
This command is used to determine the data specification for each axis.
First the user is prompted to choose the axis to which to apply the settings (x, y, z or c (the latter standing for ‘colour’).
Following this, one of four possible data modes must be chosen. These are:
signal_data, time, constant, sensor_number
It is the combination of these 4 data modes for 4 different axes that gives the so-called xy-display its flexibility.
Details of each data mode are:

signal_data
For the x, y and z axes this would be the most usual choice, e.g the data for the x-axis is determined by the x-coordinate of the sensor data (for the colour axis this choice would be unusual, but perfectly possible, e.g if a velocity signal is available then trajectory colour could be determined by sensor velocity).
If this mode is chosen then the user must then reply to the following prompt:
'Generate signal list from trajectory names?'
Replying with ‘y’ (yes) is the simplest and most usual case. In this case the user is then prompted for the suffix to add to the trajectory names to generate the signal names.
For example, if trajectories have been defined with names ‘tt_’, ‘tm_’, and ‘tb_’ then the signals used for the x-axis will be ‘tt_x’, ‘tm_x’ and ‘tb_x’ if ‘x’ is specified as the suffix.
The following note is mainly relevant for 3D EMA data:

Assume that the sensor coordinates in an EMA system are defined as ‘x’ for lateral, ‘y’ for anterior-posterior, and ‘z’ for vertical position respectively. In order to set up a traditional sagittal display one might think that the obvious way to proceed is to map sensor ‘y’ coordinates to the graphics x-axis, and ‘z’ coordinates to the graphics ‘y’ axis (mutatis mutandis for coronal and axial views). In fact, it is generally better to set up all views of the data with a complete - **and identical** - 3D specification (e.g x to x, y to y and z to z), and then adjust the way each axes system is viewed to get e.g sagittal, coronal or 3D views. There are currently no interactive commands for changing the views of the axes in the xy figure: this has to be done by calling the function mt_sxyv.
There are two reasons for proceeding in this way: Firstly, for any 3D data (not just EMA) it is probably easier to write a command file if all axes in the xy figure have exactly the same specification (if the data is 3D it is probably better to treat it as such; the view can be changed freely). Secondly, in the special case of 3D EMA, setting up the display of sensor orientation can be extremely confusing if the scheme suggested here is not followed (this is discussed further below).

time
This specification is often used for the colour axis. Colour of each trajectory is modulated according to relative time in the trajectory; this means that trajectories for different sensors have the same colour at the same time instant, making it for example easier to follow the phasing relationships between sensors in the trajectory display.
‘time’ can, however, also be used as a specification for any of the 3 spatial axes (x, y, or z). For example, setting the graphic x-axis to ‘time’ and the graphic y-axis to the sensor vertical coordinates gives an alternative to the standard time-wave display in the f(t) figure (in particular, any number of sensors can be collated  in one panel, which is not possible in the f(t) figure.

constant
Using this specification means that the position of the trajectory on one of the spatial axes, or the colour of the trajectory can be determined by any expression that evaluates to a scalar value.
An example would be to use the trial number (in combination with axes ‘hold’ mode) so that data from different trials can be overlaid, but distinguished either by colour or position on one of the axes. This can either be done by hand, i.e when prompted for ‘constant expression’ give a value like ‘5', ‘8', ‘15' etc, and change this before every new plot, or, more elegantly by specifying an appropriate function that will evaluate to an appropriate value, e.g for the present case mt_gtrid(‘trial_number’) ???check double quotes??? Since the expression is evaluated each time a trajectory is drawn the constant expression only has to be specified once.
As a further example, it would be quite easy to write one’s own function to extract the repetition number of an item if this is contained in the label of each trial or segment.

sensor_number
This can be used as a way of making it easier to distinguish the trajectories of different sensors (i.e by colour, or position on one of the spatial axes). ‘sensor_number’ here simply means the the position in the list of trajectories given as ‘trajectory_names’ in the axes_settings described above (i.e it has nothing to do with the sensor number in the original data acquistion system)

====================================

The remaining properties in the setup of the xy figure concern the so-called sub-cut display. This will be documented when the usage of sub-cuts and markers has been homogenized.