Introduction

The TSlib toolbox contains a suite of MATLAB functions and Graphical (GUI) tools for organizing and analyzing collections of time-stamped experimental data. Time-stamped data (tsdata) is a two columned matrix in which the first column is a time stamp and the second column is an integer event code that signifies what event occurred at the given time. The time stamp is a positive number in arbitrary units. The event code numbers must be positive integers and may range from 11-99999. Variables names (event code names) can be associated with the event code numbers to make it easier to refer to them. One file of tsdata is referred to as a session. Groups of sessions are associated with a subject and together these subjects and their associated sessions form an experiment.

Each session of tsdata gets loaded from a file that is created by ones experimental system. These files contain the tsdata and additional information about that session such as which subject was run, what time it was run, etc. The TS system supports a number of file formats. In conjunction with TSlib, a system has been developed that creates session files from Med-PC programs (ref).

Included with this manual is a folder called Example that contains a sample set of four session files for a contrived experiment that will serve for a detailed example (see “An Example Script”). The sessions are for two successive days with two subjects. In this example, experimental chambers consist of two feeding stations into which subjects (mice) can poke their nose. A sensor detects the nose pokes and they are reported as the event code names PokeOn1 (number 1011) and PokeOn2 (number 1012). The sensor reports when they exit the stations with the codes PokeOff1 (1001) and PokeOff2 (1002). When a feeding station feeds, it is recorded as codes Feed1 (21) and Feed2 (22). Each station has a light that can be illuminated, indicated by codes LightOn1 (41) and LightOn2 (42), and turned off, indicated by codes LightOff1 (31) or LightOff2 (32). There is a tone generator that can be turned on, indicated by event code ToneOn (61), and turned off, indicated by ToneOff (51). There is a white noise generator that can be turned on, indicated by event code WhiteNoiseOn (81), and turned off, indicated by WhiteNoiseOff (71). Other used event codes are StartTrial1 (111), which indicates the start of tone trials, StartTrial2 (112), which indicates the start of white noise trials, EndTrial (121), which indicates that a trial is over, and StartSession (115) and EndSession (125), which indicate the start and end of the session.

In this made up example, there are two types of trials and following each trial there is an inter-trial interval. The trials are either tone trials (indicated to the subject by the presence of the tone) or they are white noise trials (indicated by the presence of the white noise). The lights in the feeding stations go on and off periodically. If a trial is active, the light is on and the subject has its nose in the station, then the subject may be fed. When a feeding occurs, the associated light turns off. The likelihood of being fed is higher in station 1 during tone trials and in station 2 during white noise trials.

Listing 1 shows the beginning of the tsdata for the first session for subject one of the example data. The tsdata itself it contained in the second and third rows. The first column shown is simply the associated row numbers and would not appear in the tsdata. The event code names in the fourth column give the variable names associated with the event code numbers and are for purposes of this manual only. The tsdata in Listing 1 can be read as:

At time unit 1, event code number 115 occurred. The event code name given to 115 is StartSession. At time unit 80, event code number 42 (LightOn2) occurred. The first trial started at time unit 201, indicated by event code number 112 (StartTrial2). In our example, the time units are in seconds.

Row #	Time	Event Code #	Event Code Name
1:	1	115	StartSession
2:	80	42	LightOn2
3:	87	32	LightOff2
4:	130	42	LightOn2
5:	132	32	LightOff2
6:	151	41	LightOn1
7:	168	42	LightOn2
8:	169	32	LightOff2

9:	174	42	LightOn2
10:	175	31	LightOff1
11:	185	32	LightOff2
12:	188	42	LightOn2
13:	201	112	StartTrial2
14:	201	81	WNoiseOn
15:	216	1012	PokeOn2
16:	216	22	Feed2
17:	216	32	LightOff2
18:	226	1002	PokeOff2
19:	226	41	LightOn1
20:	228	1012	PokeOn2
21:	229	1002	PokeOff2
22:	230	1012	PokeOn2
23:	232	1002	PokeOff2
24:	235	1011	PokeOn1
25:	257	21	Feed1
26:	257	31	LightOff1
27:	260	1001	PokeOff1
28:	263	1012	PokeOn2
29:	267	1002	PokeOff2
30:	269	1011	PokeOn1
31:	271	1001	PokeOff1
32:	271	42	LightOn2
33:	272	1011	PokeOn1
34:	278	41	LightOn1
35:	284	21	Feed1
36:	284	31	LightOff1
37:	289	1001	PokeOff1
38:	291	1012	PokeOn2
39:	291	32	LightOff2
40:	292	42	LightOn2
41:	293	22	Feed2
42:	293	32	LightOff2
43:	294	1002	PokeOff2
44:	296	42	LightOn2
45:	302	1012	PokeOn2
46:	304	1002	PokeOff2
47:	307	32	LightOff2
48:	308	1011	PokeOn1
49:	316	1001	PokeOff1
50:	321	41	LightOn1
51:	323	1011	PokeOn1
52:	324	42	LightOn2
53:	327	31	LightOff1
54:	332	71	WNoiseOff
55:	332	121	EndTrial

Listing 1. First trial of an example session

Installing TSlib

Stylistic Notes

In this manual, a few conventions will be adhered to.

All underlined headings are TSlib functions.

The Experiment Structure

The TSlib functions and GUI always apply to a global MATLAB structure called Experiment. The Experiment structure contains all information about the subjects, the sessions of time-stamped data, a number of settings, a number of automatically computed statistics and any user-computed statistics. It is hierarchically arranged using nested structure arrays.

The Experiment level

Each field at the Experiment level is of the form Experiment.<fieldname>. The top level of the Experiment structure contains the following fields:

Notes	- Text notes about experiment (a string)
Name	- Experiment name (a string)
Id	- Experiment ID number (an integer)
StartDate	- Date of earliest loaded session in experiment (a string in "dd/mmm/yyyy" format)
EndDate	- Date of most recently loaded session in experiment (a string in "dd/mmm/yyyy" format)
Lab	- Lab name (a string)
Species	- Species of subjects (a string)
NumSubjects	- Number of subjects (an integer)
Subjects	- Ids of subjects used in experiment (an array of integers)
Subject(*i*)	- Structure containing information and sessions for each subject (see below)
Info	- Structure containing settings (see below)
EventCodes	- Structure of event codes (see below)
Trial<*usertrial*>	- Any number of trial definitions.
<*userstat*>	- Any number of user-computed statistics at the Experiment level

The Experiment.Subject(i) level

Each field at the Subject(*i*) level is of the form Experiment.Subject(*i*).<fieldname>. The *i* indicates which subject (by index) is being referred to. The Experiment.Subject(*i*) level contains the following fields:

Notes	- Text notes about subject (a string)
Id	- Id number of subject (an integer)
Strain	- Strain of subject (a string)
Sex	- Sex of subject (a string, typically 'M' or 'F')
Birthdate	- Birth date of subject (a string in "dd/mmm/yyyy" format)
ArrivalDate	- Date subject arrived at lab (a string in "dd/mmm/yyyy" format)
ArrivalWeight	- Weight of subject (a real number, typically in grams)
NumSessions	- Number of recorded sessions for this subject (an integer)
Session()	- Array of session structures containing all session information
<userstat>	- Any number user-computed statistics at the subject level

The Experiment.Subject().Session() level

Each field at the Session() level is of the form Experiment.Subject().Session().<fieldname>. The Experiment.Subject().Session() level contains the following fields:

Notes	- Notes about this session
Weight	- Weight of subject before this session
Date	- Date of this session in "dd/mm/yyyy" format
StartTime	- Time that session started in "hh:mm:ss AM/PM" format
Duration	- Duration of session in "hh:mm:ss" format
MATLABStartDate	- Start of session in MATLAB time format
MATLABEndDate	- End of session in MATLAB time format
Experiment	- Number given to this experiment for this session
Phase	- Phase number assigned to this session
Box	- Experiment box used to run this session
Program	- Text containing the program used to run this session
XXX TSdata!!!	
Trial<trialname>	- Structure containing information on trial type <trialname>
<userstat>	- Any of number user-computed statistics

The Notes field can contain text (a string) describing this session.

The Weight field contains the weight in grams of the subject before this session was run.

The Date field contains the date (a string in dd/mm/yyyy format) that this session was run.

The StartTime field contains the time (in "hh:mm:ss AM/PM" format) that this session was started.

The Duration field contains the duration ("in hh:mm:ss" format) of this session.

The MATLABStartDate field contains the date that this session started in MATLAB time format (Number of days since January 1, 0000).

The MATLABStartDate field contains the date that this session ended in MATLAB time format (Number of days since January 1, 0000).

The Experiment field contains the id number (an integer) given to this experiment when this session was run.

The Phase field contains the phase number (an integer) assigned to this session.

The Box field contains the number of the experiment box / apparatus(an integer) that was used to run this session.

The Program field contains the text (a string) of the program that was used to run this session.

The Trial<trialname> field is a structure containing information regarding the trial defined as <trialname>.

The <userstat> field contains any user-computed statistics at the session level (i.e. statistics computed on a session or over trials). There may be any number of such fields.

The Experiment.Subject().Session().Trial<trialname> level

Each field at the Trial<trialname> level is of the form Experiment.Subject().Session().Trial<trialname>.<fieldname>. The Experiment.Subject().Session().Trial<trialname> level contains the following fields:

NumTrials - The number of trials in this session
Trial() - Array of trial structures

The NumTrials field contains the number (an integer) of trials that exist in this session.

The Trial() field is an array of structures containing all of the information for each trial in this session.

The Experiment.Subject().Session().Trial<trialname>.Trial() level

Each field at the Trial(k)level is of the form Experiment.Subject().Session().Trial<trialname>.Trial().<fieldname>
The Experiment.Subject().Session().Trial<trialname>.Tiral() field level contains the following fields:

StartTime - Start time of this trial in defined units from start of session
EndTime - End time of this trial in defined units from start of session
Duration - Duration of this trial in defined units
sloc - Starting location (row number) of this trial
eloc - Ending location (row number) of this trial
<userstat> - Any number of user-computed statistics

The StartTime field contains the start time of this trial (an integer) relative to the start of the session.

The EndTime field contains the ending time of this trial (an integer) relative to the start of the session.

The Duration field contains the duration (an integer) of this trial.

The sloc field contains the row number in this session where this trial starts.

The eloc field contains the row number in this session where this trial ends.

The <userstat> field contains any user-computed statistics at the trial level. There may be any number of such fields.

The Experiment.Info level

The fields stored under Info are primarily for internal use by the TSlib system. It is not recommended to set these fields directly.

Each field at the Info level is of the form Experiment.Info.<fieldname>. The Experiment.Info level contains the following fields:

ActiveSubjects - Active subjects (default [1 TSend])
ActiveSessions - Active sessions (default [1 TSend])
ActivePhases - Active phases (default [1 TSend])
ActiveDates - Active dates (default [1 TSend])

ActiveTrials	- Active trials (default [1 TSend])
ActiveTrial	- Active trial definition
ActiveData	- Active tsdata for analysis
OverWriteMode	- Overwrite mode (true (1) or false (0) – default is 0)
FilesLoaded	- List of files loaded into Experiment by TSloadsessions
TSLibVersion	- Version of TSlib used to create this Experiment Structure
InputTimeUnit	- Time unit of loaded session files in seconds
OutputTimeUnit	- Time unit for internal (stored) session data in seconds
LoadFunction	- Function used to load in session files
FilePrefix	- Prefix String used to denote session files
FileExtension	- Suffix String used to denote session files

The ActiveSubjects, ActiveSessions, ActivePhases, and ActiveTrials fields all contain a four-columned matrix that gives a high and low range for which data to consider when doing analyses. ActiveDates uses a two-columned matrix. See TSlimit for more information on how to set these and how the analyses are affected.

The ActiveTrial is a string the gives the name of the trial definition that will be used when doing analyses on trials. Note that all trial names get the word "Trial" pre-pended onto them by the TS system.

The ActiveData contains the name (a string) of the data field that will be used when doing analyses on sessions. By default, this field is TSData which is where TSloadsessions places the session data.

The OverWriteMode field is either true (1) or false (0) and it determines whether or not various TSlib functions overwrite previously computed statistics and whether or not previously loaded session files are reloaded.

The FilesLoaded field contains a cell array of filenames of the session files that have been loaded into the Experiment structure. If the OverWriteMode field is false then this list is used to only load in sessions that have not been previously loaded.

The TSLibVersion field contains a string indicating the version of TSlib that was used to create this Experiment structure.

The InputTimeUnit field gives the time unit in seconds used by the session files that are read in. For example, if this were .02 it would indicate that the time stamps are in 50ths of a second. The default for this is 1, indicating that the time-stamps in the read in session files are in seconds.

The OutputTimeUnit field gives the time unit in seconds of time stamps used in the Experiment structure. When session data is read in, the stamps are converted to these units. The default for this is 1, indicating that the time-stamps in the saved session files are in seconds.

The LoadFunction field contains the function as a string or a function handle that will be used by TSloadsessions to load in session data files. The default for this is "TSloadstdtab", which is the standard tab delimited format supported by TSlib. See the section on Session File Formats for more information on supported formats and how to write loaders for your own file formats.

The FilePrefix field can contain a string that specifies a prefix used in files that should be loaded by TSloadsessions. TSloadsessions will only read in files beginning with this string. If this is '', then all files in the specified folder will attempt to be read and this is the default setting.

The FileExtension field can contain a string that specifies an extension used in files that should be loaded by TSloadsessions. TSloadsessions will only read in files ending with this string. If this is '', then all files in the specified folder will attempt to be read and this is the default setting.

Using scripts and the Experiment Browser

The TSlib functions allows one to load a collection of session tsdata files, organize them into the hierarchical Experiment Structure, analyze the data, and to store the results back into the Experiment Structure in the appropriate context. These results, or "statistics", can be further combined and analyzed.

The most convenient way to perform all these operations is using an M-file script to queue up a series of TSlib functions. Using this scheme, it is possible to rebuild your entire Experiment and analysis with a single command. This makes what would be an otherwise tedious process very simple and easily reproducible.

In the Example folder is a file called "Manual_Example_100.m". This file demonstrates the use of an M-file to store sequences of TS commands. See the section called "An Example Script" for a full listing and explanation of this script. The results of running this file have been stored as Manual.Example_100.mat. To load this example Experiment Structure, use the TSlib function TSloadexperiment('Manual_Example_100');, with the Example folder being the current MATLAB directory.

Once you have loaded this experiment, you can look at the fields directly from MATLAB. For example, typing in "Experiment.Species" will return back "Mouse".

TSlib also provides powerful GUI tools to easily browse and visualize the Experiment Structure and the data. To bring up the browser, use the TSlib function TSexperimentbrowser; This GUI allows you to easily walk through the Experiment structure, edit the structure, and perform a number of advanced functions. For a detailed explanation, see the section called "The Experiment Browser".

Working with Event Codes

All event codes in tsdata are integers between 11-99999. When dealing with tsdata, it can be cumbersome remembering the specific code numbers and therefore it is useful to have variable names such as "StartTrial" that can be used to refer to these numbers. TSlib supports the use of named event codes.

It is very good practice to maintain some consistency within a lab regarding the event codes. If this is done, then there is a good chance that one could interpret raw experiment files years later if the need arose. Each Experiment structure maintains its own set of event codes so that when an Experiment structure is shared, all of its named event codes travel with it. TSlib supports the importing and exporting of event codes from files. Event code files are simply text files of the form:

```
<event-code-name1> = <event-code1>;
<event-code-name2> = <event-code2>;
```

That is, it is simply a series of variable definition for use within MATLAB. For example, listing 2 shows the event code file used in the manual example experiment:

```
StartTrial1 = 111;
StartTrial2 = 112;
StartSession = 115;
EndTrial = 121;
EndSession = 125;
Feed1 = 00021;
Feed2 = 00022;
LightOff1 = 00031;
LightOff2 = 00032;
```

LightOn1 = 00041;
LightOn2 = 00042;
ToneOff = 00051;
ToneOn = 00061;
WNoiseOff = 00071;
WNoiseOn = 00081;
PokeOff1 = 01001;
PokeOff2 = 01002;
PokeOn1 = 01011;
PokeOn2 = 01012;

Listing 2, Manual example event codes (ManualCodes.txt)

TSimporteventcodes

TSimporteventcodes; % Import event codes from a located file

TSimporteventcodes ('ManualCodes.txt'); % Import event codes from the file 'ManualCodes.txt'

This function loads in a text file that contains a sequence of event code assignments and stores it in the Experiment structure.

TSexporteventcodes

TSexporteventcodes ('OurCodes.txt'); % Save the Experiment structure event codes to a file called "OurCodes.txt".

This function saves the event codes defined in the current Experiment structure into a text file.

TSsetdefaulteventcodes

TSsetdefaulteventcodes('OurCodes.txt'); % Load OurCodes.txt as the default code set.

TSsetdefaulteventcodes; % Prompt user to supply a filename or use Current Codeset as default

This function is used to set a default codeset which is used in the event that there is no Experiment Structure defined, or if the current Experiment does not have any codes associated with it. This is stored into MATLAB's preferences. This may be useful if you need to perform testing on data cleanup or analysis routines but do not have an Experiment currently loaded.

TSaddeventcodes

TSaddeventcodes('Feed3',23); % Add an event code called Feed3 with the value 23.

TSaddeventcodes('Feed3',23,'Feed4',24); % Add an event code called Feed3 with the value 23 and an event code called Feed4 with value 24.

This function allows one to add one or more event codes to the Experiment structure.

TSrmeventcodes

TSrmeventcodes('Feed3'); % Remove the event code called Feed3.

TSrmeventcodes('Feed3','Feed4',); % Remove the event codes Feed3 and Feed4.

This function allows one to remove one or more event codes from the Experiment structure.

TSdeclareeventcodes

TSdeclareeventcodes; % Allow the use of event code names in a function

This function declares the event code definitions as global variables, and also puts them within the workspace of the calling function and the base workspace. This allows you to use the codes by name rather than referring to them by number. To allow for this within your own function, simply include this function call at the top of your function. Once this is done, you may refer to all event codes by name.

S = TSdeclareventcodes; % Get the event code structure as an output.

When an output is used with TSdeclareeventcodes, it returns the current code structure, which may be Experiment.EventCodes or may be the default code set stored in preferences. This useful if your function needs a list of all the event code names, which you can get simply by using the fieldnames function on the returned structure. For example, TSrastergui uses this method to populate the event code lists.

Creating, Saving, and Loading Experiment Structures

TSlib provides functions for creating, saving, and loading Experiment structures. The Experiment Browser also allows for these operations.

TSinitexperiment

TSinitexperiment('Manual_Example_100', 100, [101 102]); % Initialize a new experiment called Manual_Example_100. Its id number is 100, and it has 2 subjects with ids 101 and 102.

TSinitexperiment('Manual_Example_100', 100, [101 102], 'Mouse', 'King'); % Same as above example, this additionally specifies the species as "Mouse", and the lab as "King".

This function creates a new Experiment structure. When created, each experiment must be given a name (a string), an identifying experiment number (an integer) and a list of subject ids (an array of integers) for the subjects participating in this experiment.

Optionally, one may give two more string arguments to TSinitExperiment that specify the species of the subjects and the lab that this experiment was conducted in.

It is recommended that the Experiment name contains no spaces and contains the experiment id number.

GUI Equivalent: To create a new Experiment structure from the Experiment Browser, select **New Experiment** (ctrl-N) from the file menu. You will then be able to enter all of the needed fields.

TSsaveexperiment

TSsaveexperiment; % Save the experiment structure under its current name

TSsaveexperiment('New_Example_100');

TSsaveexperiment saves the Experiment structure to a .mat file. If no filename is specified, it will save it under its current name (specified in TSinitexperiment). If a file name is specified, the structure will be saved under this new name.

GUI Equivalent: To save an Experiment structure from the Experiment Browser, select **Save Experiment** (ctrl-S) from the file menu. To save the structure under a new name, select **Save As** (ctrl-A) and you will be prompted for the new name.

TSloadexperiment

TSloadexperiment;

TSloadexperiment('Manual_Example_100');

TSloadexperiment loads in a saved Experiment .mat file. If given no arguments, it will bring up a dialog box to locate the file. Otherwise, it will load the file given.

GUI Equivalent: To load an Experiment structure from the Experiment Browser, select **Load Experiment** (ctrl-L) from the file menu.

Loading Session Data

TSlib can load in files that contain a session of time-stamped data and store them into the Experiment structure. The function TSsetloadparameters is used to set a number parameters that determine how files are loaded and the function TSloadsessions actually does the loading.

TSsetloadparameters

Tssetloadparameters('InputTimeUnit',.02); % Indicates that the time stamps in the loaded data are in 50ths of a second.

Tssetloadparameters('OutputTimeUnit',60); % Indicates that the time stamps should be converted to minutes in the Experiment structure.

Tssetloadparameters('LoadFunction','loadACME'); % Indicates that when data is loaded, the function called loadACME should be used.

Tssetloadparameters('FilePrefix','!'); % Indicates that when data is loaded, only files beginning with should be used.

Tssetloadparameters('FileExtension','mpc'); % Indicates that when data is loaded, only files with the .mpc extension should be used.

Tssetloadparameters('InputTimeUnit',.02, 'OutputTimeUnit',60); % Indicates that the time stamps in the loaded data are in 50ths of a second and should be converted to minutes.

When TSloadsessions (see below) is used to load in session data, there are a number of parameters that is uses to decide how the data is loaded. TSsetloadparameters allows you to set these various parameters. Each parameter has a name and TSsetloadparameters allows you to specify pairs of parameter names and there value. The parameter names are case-insensitive. The supported parameters are as follows:

InputTimeUnit: This specifies the unit of time used in the loaded files in seconds. For example, if the loaded files are time-stamped in 50ths of a second, then the value specified here should be .02. If no value is specified, it is

assumed that the loaded data is time-stamped in seconds. Note, however, that this value is only used when converting time-stamps to other units with the OutputTimeUnit parameter.

OutputTimeUnit: This specifies the unit of time in seconds that should be used when storing tsdata in the Experiment structure. For example, if you want to work with the data in minutes, then the value specified here should be 60. If no value is specified, the loaded data will be converted to seconds.

LoadFunction: When TSloadsessions loads in session files, it uses a helper function to actually load and parse the individual files. This parameter specifies the function that should be used as either a string giving the name of the function or a function handle. The default for this is "TSloadstdtab", which is the standard tab delimited format supported by TSlib. See the section on Session File Formats for more information on supported formats and how to write loaders for your own file formats.

FilePrefix: TSloadsessions loads in session files from a specified folder. It may be the case that other files are also stored in the same folder and therefore it can be useful to specify that only files with a given prefix string be loaded. TSloadsessions will only read in files beginning with this string. By default, this is set to the character '', allowing all files to be read.

FileExtension : TSloadsessions loads in session files from a specified folder. It may be the case that other files are also stored in the same folder and therefore it can be useful to specify that only files with a given extension string be loaded. TSloadsessions will only read in files ending with this string. By default, this is set to the character '', allowing all files to be read.

TSloadsessions

TSloadsessions; % Brings up a dialog to find the folder containing sessions

TSloadsessions('c:\experiment\data'); % Loads files from specified data folder

TSloadsessions loads in session data files from a specified folder using the parameters set by TSsetloadparameters. It will load in all files from this folder that start with the FilePrefix parameter and end with the FileExtension parameter, it will use the function specified by the LoadFunction parameter to do the loading, and it will convert the time stamps into the time units specified by the OutputTimeUnit parameter under the assumption that the InputTimeUnit parameter is set correctly.

TSloadsessions takes a string representing a directory from which it will load the data files. If it is not given one, it prompts the user for one with a dialog.

TSloadsessions will not reload previously loaded sessions unless the OverWriteMode mode is turned on (see TSoverwritemode).

The tsdata from the session will be stored in the TSData field under the session field.

The TSloadsessions command returns 2 arguments: result – 1 or 0, to indicate (un)successful completion and SubSes - a 2-column array with one row per loaded session and the subject's ID number in the first column and the session number in the 2nd column.

Creating Statistics

In addition to organizing ones tsdata into a MATLAB structure, TSlib provides a number of functions to flexibly perform analysis on the tsdata and collect these results under what are called statistics.

Match Codes

Fundamental to working with tsdata using TSlib is the notion of match codes. A match code is simply a vector of event codes that is meant to "match" a sequence of tsdata. The match is used to identify some codes that one wants to work with. Although tsdata contains time stamps and event codes, the match codes only apply to the event code portion.

Before giving a detailed understanding, a quick example may prove useful. Let's take the portion of tsdata shown in Listing 1 and let's take a match code of [LightOn1 LightOff1]. This match code would match the tsdata at row numbers 6 and 10, 19 and 26, 34 and 36, and 50 and 53.

The event codes in the match code vector form a sequence. A "match" occurs when the match codes can be paired up with event codes in the tsdata in the proper order. Each code in the match code must match to the tsdata but the codes in the tsdata need not be sequential. In the example above, the first match occurred at row 6 (the LightOn1 matched) and the second match occurred at row 10 (the LightOn2 matched)

Once a single match is found, the next match in the tsdata with that same match code can start with the last code used in the first match. So, for example, using the same tsdata, a match code of [PokeOn1 PokeOn1], would produces matches at rows 24 and 30, 30 and 33, 33 and 48, and 48 and 51. If a match code consists of a single event code, a match will occur at each position that code matches. For example, match code [Feed1] would produce matches at rows 25 and 35.

TSmatch

form: [match, bindings] = TSmatch(tsdata, matchcodes);

TSmatch is the TSlib function that searches tsdata for matches. This function is not intended to be directly used, but instead is called from within a number of other TSlib functions. A detailed understanding of TSmatch and how it works with match codes will aid in using TSdefinetrial, TSParse, TSapplystat, TStrialstat, TSsessionstat, and TSedit.

TSmatch searches tsdata for matches to match code patterns. It works in a fashion similar to a search engine or a regular expression engine. The first argument it takes is a two-columned array of tsdata. The second argument is either a single match code vector or a cell array of match code vectors. One can think of the match codes as being a search query that is run over the tsdata. Unlike searching a text document for a piece of text, where each match is found one at a time, TSmatch finds all matches at once and returns them all.

TSmatch returns its results in two pieces. The first piece, which we will call the match, is a vector of integers that indicates which match code matched. We will come back to this when we examine multiple match code searches. The second piece, which we will call the bindings, consists of a cell array of vectors that contain the row position of where each element in the match code matched. Using the example above with tsdata equal set as in Listing 1, we could call the following:

[match, bindings] = TSmatch(tsdata,[LightOn1 LightOff1]);

In this case, bindings{1} = [6 10], bindings{2} = [19 26], bindings{3} = [34 36], and bindings{4} = [50 53]. Because there was only one match code in this example, the match returned would be the vector [1 1 1 1] as each binding came from matching the first (and only) match code.

Multiple Match Codes

If multiple match codes are given as specified by a cell array, then each match code will be searched for parallel. It is important to understand that matches are exclusive; no 2 matches are allowed to overlap when the result is returned. As soon as one of the match codes completely matches, that match is completed and the search goes on, starting with the last code matched. For example, using the same tsdata, we could perform the following:

[match, bindings] = TSmatch(tsdata,{[PokeOn1 Feed1 PokeOff1] [PokeOn2 Feed2 PokeOff2]});

This would produce a match of [2 1 1 2], with bindings{1} = [15 16 18], bindings{2} = [24 25 27], bindings{3} = [30 35 37], and bindings{4} = [38 41 43].

With multiple match codes, a match is considered to have occurred once all event codes in a particular match code have been matched. This can sometimes produce unexpected results, as some apparent matches will not be found. For example, examine the following:

[match, bindings] = TSmatch(tsdata,{[LightOn1 LightOff1] [LightOn2 LightOff2]});

The first two matches found are for the [LightOn2 LightOff2] match code, as would be expected. The third match found, however, is also for the [LightOn2 LightOff2] match code, with the bindings occurring at rows 7 and 8. Even though the LightOn1 was found at row 6, the [LightOn2 LightOff2] match code was found before the matching LightOff1 was found. Because of this, that particular LightOn1, LightOff1 pairing will not be found by this call to TSmatch.

If two match codes simultaneously make their match, then the tie is determined by the ordering of the match codes. For example:

[match, bindings] = TSmatch(tsdata,{[LightOn1 LightOff1] [LightOn2 LightOff1]});

Results in a first match of the [LightOn1 LightOff1] at row positions 6 and 10. Reversing the order of the match codes in the case:

[match, bindings] = TSmatch(tsdata,{[LightOn2 LightOff1] [LightOn1 LightOff1]})

Results in a first match of the [LightOn2 LightOff1] at rows 2 and 10.

Using Negative Event Codes

It is often very useful to specify that a match code must <u>not</u> contain a certain code, This greatly increases the flexibility of TSmatch. To indicate that a certain event code must not appear in a certain position, use a negative sign in front of the event code. For example:

[match, bindings] = TSmatch(tsdata,{[LightOn1 -Feed1 LightOff1]});

Results in two matches with bindings{1} = [6 10] and bindings{2} = [50 53]. Note that only the positive codes are included in the bindings! Compare this to the following:

[match, bindings] = TSmatch(tsdata,{[LightOn1 LightOff1]});

Here, two additional matches are found with bindings at [19 26] and [34 36]. If one wanted to only get the LightOn1 LightOff1 combinations that occurred with a Feed1 event occurring between them, the following would be used:

[match, bindings] = TSmatch(tsdata,{[LightOn1 Feed1 LightOff1]});

Negative event codes provide a mechanism for undoing a previously matched code. For example, with the match code [[LightOn1 –Feed1 LightOff1], TSmatch starts looking from the beginning of the tsdata for the LightOn1 code. If this is found, say at row 19, it will begin looking forward for the next positive code (LightOff1) under the provision that it does not encounter a Feed1 code. When it does, in fact, find the Feed1 code (in this case at row 25), it effectively undoes the match at row 19 and begins searching again for the LightOn1 code from position

If successive negative event codes appear, then TSmatch will backtrack to the most recently matched positive code if <u>any</u> of the negative codes match. That is, the order of successive negative codes plays no role in a match. For

example, consider tsdata consisting of the codes [20 30 40 30 50 30 60] and a match code of [20 30 -40 -50 60], the 20 would match the first 20. The 30 would then match the first 30 but this would be undone when the 40 was encountered. The search for another 30 would continue until the second 30 is found. This would again be undone when the 50 is found. Searching again for a 30, it would find the third 30 and then the match would be completed with the 60.

It is useful to think of a string of negative codes as a logical or condition. For example, the match code [LightOn1 –Feed1 –Feed2 LightOff1] would find any pairing of LightOn1 and LightOff1 that does not have a Feed1 or a Feed2 event between them.

Negative match codes can also cause more than one previously matched code to be backtracked. For example, let's take the following tsdata in which the time stamps are conveniently identical to the row numbers:

1 20
2 30
3 40
4 20
5 30
6 50

And perform the following call to TSmatch:

[match, bindings] = TSmatch(tsdata,{[[20 30 -40 50]]});

In this case, we will get a match with bindings{1} = [1 5 6]. The first 20 matches, the 30 at row 2 matches but is cancelled by the 40 at row 3. the 30 re-matches at row 5 and then the 50 matches at row 6. Contrast this to the following call:

[match, bindings] = TSmatch(tsdata,{[[20 -40 30 -40 50]]});

Now we get a match with bindings{1} = [4 5 6]. In this case, the first 20 matches at row 1, the 30 matches at row 2 but is then cancelled by the 40 in row 3. This 40 also cancels the match of the 20 at row 1 and the search for a 20 resumed. The 20 re-matches at row 4, the 30 matches at row 5 and the 50 matches at row 6.

Using TSstart and TSend

It is often very useful to be able to match to the very first event code in the tsdata or the very last code, regardless of what code appears there. The special codes TSstart and TSend can be used for this purpose. As a useful example, suppose we want to find all ranges in the tsdata where the mouse had its head in the first hopper. A first approximation to this would be to simply use the match code [PokeOn1 PokeOff1[. This, however, could miss the case where the mouse had its head in the hopper before the session started and/or it had its head in the hopper when the session ended. The cure for this is to use the match codes {[PokeOn1 PokeOff1], [TSstart PokeOff1] [PokeOn1 TSend]}.

Using the 's' flag

Normally, TSmatch determines that a match has been completed once all of its event codes have matched. If two match codes both match their last event code at the same row, then this tie is broken by the order in which the match codes appear in the match code cell array. A flag is available which will use match in which the <u>first</u> event code matched first, regardless of when the match completed. This flag is given as the last element of the match code cell array as the character 's'. For example, assume the following tsdata:

1 20
2 30
3 40

4 50

And perform the following call to TSmatch:

[match, bindings] = TSmatch(tsdata, {[20 50] [30 40]});

As expected, the second match code is matched at rows 2 and 3. Now, add the 's' flag, as in:

[match, bindings] = TSmatch(tsdata, {[20 50] [30 40], 's'});

In this case, the first match code matches as the 20 was the first code found. When using the 's' flag, ties occur when two match codes match and contain the same starting code. In this case, ties are again broken by deferring to the order of the match codes.

Examples

Using combinations of multiple match codes, negative event codes, and the 's' flag allow for a large variety match specifications. Here are a number of examples to demonstrate this variety.

TSmatch(tsdata, {[LightOn1 Feed1 –Feed1 LightOff1] }); % Find the ranges where light 1 went on and off and the last Feeding that occurred in these ranges.

XXX More examples

TSdefinetrial

form: TSdefinetrial(trialname, matchcodes);

One of the uses of match codes is specify ranges of tsdata that are commonly referred to as "trials." Usually these ranges will correspond to ones traditional concept of an experimental trial, but in fact, a trial definition is simply some way of segmenting tsdata into mutually exclusive ranges. Using these definitions, statistics can be applied to the trials separately (notably with the use of the TStrialstat.

Often trials are identifiable by specific start and end conditions, conditions that also specify the trials to the subjects. For example, a trial might start with a ToneOn event and end with a Feed1 or a Feed2 event. In this case, a suitable trial definition would consist of the match codes {[ToneOn Feed1] and [ToneOn Feed2]}. To create a trial definition, use the TSdefinetrial function. This function takes a name for the created trial definition and a cell array of match codes that define the trials. For example, we might define a trial based on the above match codes as:

TSdefinetrial('Feeding', {[ToneOn Feed1] [ToneOn Feed2]}); % Create a trial type called "Feeding"

Technical note: When this function is called, the trial definition will be added to the Experiment structure at the top level of the structure as a field with the word "Trial" pre-pended to the given trial name. The above example, when used in the context of the event codes defined by Listing 1 will result in Experiment.TrialFeeding = {[61 21] [61 22]}.

Trial definitions may use any legitimate match codes, however, the trial is only defined by the range specified between the first matching event code and the last. Examine the following trial definition:

TSdefinetrial('NoFeeding', {[LightOn1 –Feed1 LightOff1]}); % Create a trial type called "NoFeeding"

This defines trials that begin with a LightOn1 and end with a LightOff1 but do not have any Feed1 events occurring within them.

In general, it is very good practice to annotate ones tsdata with as many codes as may prove useful later on (see Kav ref for more information on this). For example, if all trials start with a tone going on and end with it going off, and some trials are considered "Probe Trials", then it is good practice to annotate these types of trials directly in the tsdata from the experimental setup as opposed to trying to determine which trials are probe trials post facto. In this case, if the experiment generated probe trials, it could be annotated with a special StartProbeTrial code as opposed to a StartTrial code. One can greatly simplify the process of trial definitions and statistical analysis by adding these codes in when the experiment is run.

In addition to defining a trial, TSdefinetrial also makes the newly defined trial the currently active trial (see TSsettrial).

TStrialstat

FORM: TStrialstat(newstat, mcode)

TStrialstat applies a specified function to each trial range (as defined by the active trial) within the tsdata in the Experiment structure and stores the result under a new name at the trial level in the Experiment hierarchy. TStrialstat requires the name for the new statistic (*newstat*) and matlab code that will be executed to produce the newstat. In addition, TStrialstat stores information about the number of trials found, and the start time, the end time, and the duration of each trial.

To demonstrate TStrialstat, we will use the example data in the folder included with this manual. We will focus on just the data shown in Listing 1.We will create a statistic that gives the number of feedings that occur in each trial. To start, we define a function that takes tsdata and a vector of event codes and returns the total count of these event codes. This function, which we will call countevents, can be defined as follows:

```
function result = countevents(tsdata,eventcodes)

result = 0;
for x = eventcodes
   result = result + sum(tsdata(:,2)==x);
end;
```

Now, we define a trial as tsdata beginning with StartTrial1 or Starttrial2 and ending with EndTrial.

TSdefinetrial('Both',[StartTrial1, EndTrial], [StartTrial2, EndTrial]);

We can now make our call to TStrialstat as follows:

TStrialstat('feedings', @countevents, [Feed1 Feed2]);

What this will do is segment each session of data (the tsdata) for each subject (see TSlimit on how to limit what data is used) based on the currently active trial (in this case TrialBoth), apply the countevents function to it along with the parameter [Feed1 Feed2], and store each trials results in the proper place in the Experiment Structure Hierarchy. If we now look at the location corresponding to the data from Listing 1, we see that:

```
>> Experiment.Subject(1).Session(1).TrialBoth.Trial(1)
ans =
   StartTime: 201.00
     EndTime: 332.00
    Duration: 131.00
        sloc: 13.00
```

```
    eloc: 55.00
feedings: 4.00
```

This tells us all the information computed for Subject 1, Session 1, and Trial 1 of the trial type TrialBoth . Note that our computed statistic, feedings, has been set to 4, which corresponds to the four feedings that occur in the data shown in Listing 1.

We will see how to compute this same statistic as well as many others using the function TSparse.

TSsessionstat

FORM: TSsessionstat(statname, funhand, arg1, arg2...)

TSsessionstat has the same form as TStrialstat and it functions in a very similar way. Instead of passing segments of the session tsdata (as determined by the trial definition), TSsessionstat passes the entire session tsdata to the function given by *funhand* and stores the results at the session level of the Experiment hierarchy.

Using the same function given in the TStrialstat example, could call the following:

TSsessionstat('feedings_ses', @countevents, [Feed1 Feed2]);

Now if we look at the location in the Experiment hierarchy for the first session of Subject we see that:

```
>> Experiment.Subject(1).Session(1).feedings_ses
ans =
    18.00
```

This tells us that there were 18 feeding events in Subject 1's first Session data.

TSsessionstat(*statname*, *funhand*, *arg1, arg2, …*) is functionally equivalent to TSapplystat(*statname*, Experiment.Info.ActiveData, *funhand arg1, arg2, …*). TSsessionstat should be used, however, as it is more efficient.

TSapplystat

form: TSapplystat(newstate, usestats, matlabcode)

TSapplystat applies a specified function to any statistic (or statistics) in the Experiment structure and stores the result under a new name at the same level in the Experiment hierarchy. TSapplystat requires the name for the new statistic (*statname*), the name of the statistic(s) to which the function will be applied (*usestats*), a function handle for the function that will be applied (*funhand*), and any additional arguments that will be passed to *funhand*.

If *usestats* is a single string, specified function will be passed *usestat* as its first argument and any remaining arguments that were given as *arg1*, *arg2*, etc... If *usestats* is a cell array of statistic names, each one will be passed in order to the given function, followed by the additional arguments.

Statname can also be either a string or a cell array or strings. If multiple strings are passed, then the extra fields are used to store additional outputs from the function. It is an error to ask for more outputs than *funhandle* provides.

If two single quotes ('') are used for *statname*, then the function will be applied, but no new output statistic will be created. This is VERY useful for applying procedures that create graphs of data. You can also simply leave out statname and the same behavior will result.

If *statname* and *usestat* are the same, TSapplystat will replace *usestat* with the computed statistic. Unless space is a major concern, however, we discourage this, because there is substantial benefit to keeping a record of the

computations made in the course of the analysis. If you overwrite the old statistic, then you don't have a complete record of the intermediate values used.

TSapplystat('TSdataclean', 'TSdata', @CleanupRoutine); %Applies CleanupRoutine, with TSdata as only argument, and storing result in TSdataclean;

TSapplystat('SesHistogram', 'TSdataclean', @HistogramFunction, 50, .05); %Applies HistogramFunction, with TSdata as first argument, and 50 and .05 passed as 2nd and 3rd argument, every time.

TSapplystat({'SesHistMax', 'SesHistMaxIdx'}, 'SesHistogram', @Max); %Applies Max function to SesHistogram, storing first output as SesHistMax, and second output as SesHistMaxIdx.

TSapplystat('',SesHistogram, @HistogramPlottingFunction); %Applies HistogramPlottingFunction function with SesHistogram as input, and with no outputs.

TSparse

form: TSparse(tsdata, mcode, matchcodes);

form: TSparse(tsdata, funhand, matchcodes);

TSparse is a function that allows one to flexibly extract information from tsdata. It calls TSmatch and is usually used in the context of either TStrialstat and TSsessionstat.

TSparse's first argument is tsdata (which is usually passed to it automatically by either TStrialstat or TSsessionstat). Its second argument (mcode or funhand) is either a piece of MATLAB code or it is a handle to a MATLAB function. The third argument is a cell array of match codes.

TSparse first uses TSmatch with the tsdata it is passed and the given match codes. As discussed in the section on TSmatch, this produces a match vector (which gives each match code that matched) and a cell array of bindings (which gives the rows where the matches occurred). From this, TSparse will make either multiple calls to the function handle given by funhand, or it will evaluate the mcode multiple times, once for each match found. The results from these calls will be combined and returned by TSparse.

Using mcode

When passing TSparse mcode (MATLAB code), each result should be stored to a vector named "result". Each result will get appended end to end to form the final result, so therefore each result must be of the same width. Within 'the mcode there are a few variables with special meaning. The variable 'match' gives the number of which match code matched. The vector 'time' contains the time stamp for each event code that matched. Therefore, time(x) is the time stamp associated with the x'th event code from the matching match code (note that negative event codes to not produce a time). The variables 'starttime' and 'endtime' give the starting time and the stopping time of the tsdata.

Let's look at an example of using TSparse with TSsessionstat using the manual example data. Let's say you want to determine the duration of time that the mice had their head in hopper 1 for each session. You can call TSsessionstat as follows:

TSsessionstat('SesPoke1Dur', @TSparse, 'result = time(2)-time(1);', [PokeOn1 PokeOff1]);

Here, TSsessionstat calls TSparse with the tsdata for each session, the given mcode and the given match code. TSparse finds each binding of the match code and computes result such that it is the value of the time that the PokeOff1 occurred minus the time that the PokeOn1 occurred. This gives the duration of the poke. As each duration is computed, they are appended end to end to form the final result that is stored by TSsessionstat in SesPoke1Dur.

Now let's look at another example that demonstrates the use of the match variable and the starttime and endtime. Let's say that we want to gather for each trial, the time into the trial that each Feed1 event occurred and the time from the end of the trial that the Feed2 events occurred. We will gather this data as pairs of numbers where 1 or 2 will indicate which feeding, and the second number will give the associated time. The result will be a 2xN array for each trial. The following trial definition and call do TStrialstat will do what is needed:

TSdefinetrial('Both',[StartTrial1, EndTrial], [StartTrial2, EndTrial]);

TStrialstat('TriToneOrWhiteNoiseTimes', @TSparse, 'if (match==1) result =[1 time(1)-starttime]; else result = [2 endtime-time(1)]; end;', {[Feed1] [Feed2]});

This result in the following array for Subject 1, Session 1, and Trial 1, of our manual example (the data given in Listing 1):

```
[2  116
 1   56
 1   83
 2   39]
```

This shows us that for this first trial there were four feeding events. The first was a Feed2 event that occurred 116 seconds before the end of the trial. The second and third feeding were Feed1 events and occurred 59 and 83 seconds after the trial started. The fourth feeding was a Feed2 event and occurred 39 seconds before the end of the trial.

Often, combinations of statistics using TSparse combined with call to TSapplystat can get you a desired result. For example, We can achieve the same counting of feeding events that we achieved in the TStrialstat example without the use of an auxiliary function:

TSdefinetrial('Both',[StartTrial1, EndTrial], [StartTrial2, EndTrial]);

TStrialstat('feedingevents', @TSparse, 'result = [1];',{[Feed1] [Feed2]});

TSapplystat('feedings', 'feedingevents', @sum);

This results in the trial level field feedings being set as it was in the TStrialstat example:

```
>> Experiment.Subject(1).Session(1).TrialBoth.Trial(1)
ans =
      StartTime: 201.00
        EndTime: 332.00
       Duration: 131.00
           sloc: 13.00
           eloc: 55.00
  feedingevents: [4x1 double]
       feedings: 4.00
```

Note that this does result in an extra and perhaps unnecessary field called "feedingevents". This can be removed using the TSrmfield function.

Sometimes it is desirable to have TSparse make a match but not actually add any information to the cumulative result record. This can be done by setting result to []. In this example, we find the session times of Feed1 events that happen when Light2 is not on:

TSsessionstat('SesFeed1NoLight2', @TSparse, 'if (match==1) result =[]; else result = [time(1)]; end;', {[LightOn2 –LightOff2 Feed1] [Feed1] 's'});

Using Function Handles

In addition to supporting embedded mcode, TSparse can accept a function handle for a function that it will call for each match. When using this syntax, your function must be of the following form:

function result = MyFunction(match, time, starttime, stoptime, varargin)

Your function should return a result back, just as was the case when using mcode. It may make use of any of the parameter variables that will be passed into it. Your function should include the varargin parameter at the end in the case that future versions of TSlib pass more parameters.

TScombineover

form: TScombineover(statname, usestat, [modeflags]);

TScombineover searches for an existing statistic (*usestat*) and combines every instance of it one level up in the hierarchy (producing the statistic *statname*). By default, this combining is done by stacking each statistic, one on top of the other, and therefore for TScombineover to work correctly, each *usestat* array must have the same width. If your statistic contains several values with different widths, you can use the 'c' flag to produce a cell array (see below).

For example, consider an Experiment Structure in which there is a session level statistic named 'SesPokes1Dur'. Consider the command:

TScombineover('SubPokes1Dur', 'SesPokes1Dur')

This will produce a subject level statistic that contains all of the poke durations from all of the sessions in a single vertical array.

TScombineover allows for an optional third argument that is a string. This string may contain character "flags" that change the way TScombineover works.

Tagging data with the 't' flag

Note that when using TScombineover, you will lose information regarding where the data came from. In the above example, you will no longer know which sessions the poke durations came from. Using the tag flag 't' will instruct TScombineover to tag on an extra column to its output which contains either the trial, session, or subject that the statistic was gathered from. This extra column will be added on to the right hand side of the returned array. For example, assume you collected the session times of Feed1 events at the trial level in a statistic called "TriBothFeed1Times" so that trial 1 had the times 26, 32 and 48 and trial 2 had times 58, 90, 105, and 134, and trial 3 had times 150, and 194. Now you can execute the following:

TScombineover('SesFeed1Times', 'TriBothFeed1Times', 't');

This results in a Session level statistic called SesFeed1Times that would look like:

```
26    1
32    1
48    1
58    2
90    2
105   2
134   2
```

150	3
194	3

Merging times with the 'm' flag

If what is being combined consists of time-stamped data, then it will often be the case that one wants to "merge" the times so that the time stamps become non-decreasing and indicating of a continuously running clock. This is accomplished by using the 'm' flag. This is done by "zeroing" each piece of TSdata (subtracting the starting timestamp from all time stamps in the piece of data, so that the starting timestamp is zero), and then appending them together in succession, each time adding the ending time stamp for the previous piece to the incoming piece's timestamps. So, if there is a series of TSdata pieces found from several different trials, then using 'm' flag will combine them together such that they are timed using a clock that only runs while a trial is occurring. In other words, the time gap between each of the pieces is eliminated in the merged version.

For example, lets say we had generated a trial statistic that produced from three trials the following time stamped data.

26 3, for trial 1which lasted from

C flag (cell array), m flag (merge)*

Often statistics are computed that include a column that is the time of the event. The times are recorded relative to the field for which the statistic is created. For example, when

Two other legal modeflags are 'm' and 'c'. Using 'm' merges the times, which will make the resulting TSdata appear to be one continuous duration of time.

Combining irregular data with the 'c' flag

Using the 'c' flag combines all the instances of the combined statistic into a cell array rather than simply concatenating them. This is useful if your statistics are not TSdata, if it does not make sense to vertically concatenate them, or if they are irregularly shaped. You can then perform a statistic analysis on the values in this resulting cell array, and all possible information will be available to you. This will not cause errors no matter how irregular your data is.

Misc TSlib functions

TSsettrial

TSsettrial('Probe');% Sets the active trial to 'Probe'

TSsettrial sets the active trial for the current experiment. This trial is the trial that is operated on by TSapplystat, TScombineover, and TStrialstat, and any other functions that use the field Experiment.Info.ActiveTrial.

If no trials are defined, the active trial will be set to 'none'. If you are not operating on statistics at the trial level, then setting the active trial to "none" will make TSapplystat and TScombineover run faster since they will not need to check the trial level.

TSdefinetrial automatically calls TSsettrial, making a newly defined trial the currently active trial.

TSaddsubjects

TSaddsubjects asks the user a series a questions and using the inputs provided, automatically adds a subject to the Experiment data structure at all the necessary levels.

TSsetoverwritemode

TSsetoverwritemode(true); % Turns on overwrite mode

TSsetoverwritemode(false); % Turns off overwrite mode

When loading in data or computing statistics, TSlib functions may be asked to load in data it has already loaded or to compute statistics that it has already computed. For example, you may have used TSsessionstat on sessions 1-10. If you now load in a session 11, calling the same TSsessionstat could cause that statistic to be re-computed for sessions 1-10. If you keep all of your data files in one folder, use of TSloadsessions could cause many files to be unnecessarily reloaded.

Although this re-computation or reloading of files won't cause any damage, it can result in TSlib running much slower then need be. On the other hand, some times it may be necessary to re-compute statistics, as is the case if the statistic definition has been changed.

To provide for these cases, TSlib maintains an overwrite flag that determines whether previously computed statistics should be re-computed and whether previously loaded files should be re-loaded. If the overwrite mode is true, then all TSlib functions will re-compute their values and files will be re-loaded by TSloadsessions. If this mode is false, then the will only load in data that has not been loaded before and they will only compute statistics that have not been computed before.

The overwrite mode is set to false as its default. To turn the overwrite mode on, use the command TSsetoverwritemode(true);. To turn it back off, use the command TSsetoverwritemode(false);

TSlimit {TSgetrange has been changed to TSgetlimit}

Form: TSlimit(fieldname, limit);

Form: Example: TSlimit('all')

By default, the TSlib functions TStrialstat, TSsessionstat, TSapplystat and TScombineover operate on all available data in the Experiment structure. TSlimit places restrictions on which subjects, sessions, phases or trials are considered by these functions. Calling TSlimit('all') removes all restrictions on all fields.

When calling TSlimit to make restrictions, the *fieldname* argument specifies the name of the field to limit (either 'Subjects', 'Sessions', Phases', or 'Trials'). The *elements* argument specifies the elements within the given field that will be used for analysis.

Limiting Subjects, Sessions and Trials

Subjects, sessions, and trials are all indexed from 1 to *n*. The TSlimit function in these cases determines the indices that will be used when creating statistics. For subjects, the number *n* is the total number of subjects in the experiment. For sessions, the number *n* is the total number of sessions given a particular subject under consideration. For trials, the number *n* is the total number of trials given a particular subject and a particular session under consideration.

If *elements* is ‘all’, then all elements 1-n will be used in the given fieldname. For example, TSlimit(‘trials’,’all’) will use all trials in analysis.

The *range* can be a single number. In this case, data analysis will be restricted to a single subject, session, or trial. The number can be either a positive integer, a negative integer, a real between 0 and 1, or Inf.

If the number is a positive integer, then the index used is simply this number. For example, TSlimit(‘sessions’, 2) will only use the second session of subjects for analysis.

If the entry is Inf, then TSgetlimit (previously TSgetrange) returns only the last trial or session or subject, etc. If value >=1, it returns that value (which should be an integer). If it’s between 0 and 1, it returns the integer that falls closest to that percent.

If the number is a negative integer, then the index used is found by counting backwards from the number *n*. So, -1 would refer to the last index, -2 to the second to last, etc. For example, if subject 1 had 7 sessions and subject 2 had 10 sessions, then using TSlimit(‘sessions’,-2) will use only session 6 for subject 1 and session 9 for subject 2.

If the number is a real between 0 and 1, then it is treated as a percentage. The index then, is the closest index value that falls that percentage from 1 to *n*. For example, TSlimit(‘trials’,.5) would only use the median trial of each session analyzed.

If the number is Inf, then the array index is simply *n*, the last entry (this is equivalent to using -1).

In addition to being a single number, the *range* can be an array of numbers, the result being all the indices computed as above for a single number. For example, if there were 20 subjects, then TSlimit(‘subjects’, [,7,Inf,.7,-3]) would limit the subjects for analysis to subject 7 (from 7), 14 (from .7) 18 (from -3), and 20 (from Inf).

The *range* can also be a cell array that consists of numbers and arrays of number pairs. Just like in the single array case, each number in the cell array determines an index, and these indices determine which elements to use. If an array of two numbers is included in the cell array, then the indices included are the range between the two computed indices For example, again assuming 20 subjects, TSlimit({.7 7 Inf [2 5] [.5 .6]}) would restrict analysis to subjects [2 3 4 5 7 11 12 14 20]. 2,3,4, and 5 come from [2 5], 7 comes from 7, 11 and 12 come from [.5 .6] , 14 comes from .7, and 20 comes from Inf.

Limiting Subjects

Note that when limiting the subjects, the numbers 1—*n* are used to refer to the index of the subjects (based on their entry order) and *not* the ID numbers of the subjects. When using this ordering, all of the options explained above for limiting sessions and trials are available.

In addition, one can limit the subjects by referring to the IDs of the subjects. In this case, the *range* must be a cell array with the first element an array of Subject IDs (or a single ID), and the second element the character ‘a’, to indicate absolute mode.

Example: TSlimit(‘Subjects’,{[304 309 412],’a’});Use subjects with IDs 304, 309, and 412.

Limiting Phases

Limiting phases works in the same way that the limiting the subjects does when using the ‘a’ . That is, the phases nubers are treated nominally. For phases, the range one either be a single phase number or an array of phase numbers. Negative numbers, negative numbers, Inf, and cell arrays should not be used. One can also use TSlimit(‘phases’, ‘all’) to use all phases.

Example: TSlimit(‘Phases,{[2 17],’a’});Use phases 2 and 17.

Note: Currently, there is no support for limiting by dates.

TSsetdata

TSsetdata('myTSData');% Sets myTSData as the active data

TSsetdata sets the active tsdata for the current experiment. This is the tsdata that is operated on by TStrialstat, TSsessionstat, and any other functions that use the field Experiment.Info.ActiveData.

TSrmfield

TSrmfield('TrialProbe');% Recursively removes Trial Probe from every level of experiment

TSrmfield is used to remove a field entirely from the Experiment structure.

TSrmfield deletes every field of the given name, not just the currently active one. If a field is removed using this function, it disappears from every Subject, every Session, and every Trial, everywhere.

This function should be used with caution, as it can damage the Experiment structure to become invalid if say, TSrmfield('Session') is executed, which would not only delete every *** Finish this paragraph ***

*TSedit**

TSEdit allows the user to flexibly edit TSData by means of searching for matchcodes and either replacing these matches or adding new information around the match. For example :

TSedit(TSData , MatchcodeArrays , EditcodeArrays)

searches the TSData for instances of MatchcodeArrays and edits matched data based on instructions from EditcodeArrays. EditcodeArrays is a set of arrays of the form [Matchnum Matchpos Newcode Newlocationoffset]. Matchnum is the index of MatchcodeArrays in which edit codes should be executed if found. For instance, MatchcodeArrays may contain multiple matchcodes specified by the user, such as {[LightOn1, LightOff1] [LightOn1, Feed1, LightOff1]}. In this case, Matchnum of 1 will refer to [LightOn1. LightOff1], and 2 will refer to [LightOn1, Feed1, LightOff1]. This allows the user to flexibly specify code to be executed upon different matches. Matchpos refers to the index of the matchcode array specified by Matchnum that is to become the base of the offset of the desired data manipulation. In the above example of Matchnum 1, a Matchpos of 1 will center the offset on the time in which the LightOn1 is matched, and a Matchpos of 2 will do the same for LightOn2. NewCode is self-explanatory, as it refers to the event code that is to be inserted or replaced over the old event code. Finally, Newlocationoffset refers to the above-mentioned offset that is the time after, before, or during which the user's data manipulation will take place. An offset of Inf will replace the code at that time. If a NewCode of 0 is given with an offset of Inf, it will delete the code at that time. Consider this simple TSData using matlabcodes and singular digit timestamps:

```
[     1     00113
      2     00041
      3     00021
      4     00031
      5     00114     ]
```

The code :

TSEdit(TSData, {[LightOn1, LightOff1] [Feed1, EndSession]}, {[1, 2, LightOff2, Inf] [2, 1, PokeOn1, 1] [1 1 StartTrial, -1]})

will have the following effect on TSData:

```
[       1            00113
                 1            00111
                 2            00041
                 3            00021
                 4            01011
                 4            00032
                 5            00114        ]
```

The matched instance of LightOff1 is changed to LightOff2, the instance of PokeOn1 is inserted into the time of Feed1 + 1 (Time 4), and the instance of StartTrial is inserted into the time of LightOn1 – 1 (Time 1).

TSwaitbar

TSwaitbar('text'); % Turns on text reporting of progress

TSwaitbar ('graphics'); % Turns on graphical reporting of progress

TSwaitbar ('off'); % Turns off reporting of progress

Sometimes in can take a while to compute statistics in TSlib over all of your data and it can be useful to see how for it has to go. By default, TSlib reports its progress in the MATLAB command window. If you would like to see the progress with a graphical wait bar, use the command TSwaitbar('graphics'). To turn off reporting of progress, use the command TSwaitbar('off'). To go back to using the text reporting, use TSwaitbar('text').

Advanced Use:

TSwaitbar is an expansion upon Matlab's built-in waitbar system. It acts as an interface to a stack of waitbars, which can provide progress updates on nested processes. The interface is very easy to use and to implement for your own custom functions so that they will provide waitbar updates as well.

The functions TSapplystat, TStrialstat, TSsessionstat, and TScombineover all use the TSwaitbar interface.

If you turn on graphical waitbars, and run an ApplyStat or one of the other functions just listed, you will see several waitbars appear in a figure window, each of which indicate progress at the Subject, Session, and Trial level.

If you write a custom function for use with ApplyStat / TrialStat / etc., and it iterates over a large for loop and takes a long time to execute, it might be desirable for it to create a waitbar as well. This can be done quite easily. Simply add the following commands at the appropriate places in your function with respect to the for loop:

```
TSwaitbar('add',0,'Example Process');
TSwaitbar('inc', .01);
for x = 1:100
   ...
   ...
   TSwaitbar('update', x/100);
end
TSwaitbar('remove');
```

For more information, see help TSwaitbar.

An Example Script

```
function Manual_Example_100     % Essentially works like a script, but allows us to have helper functions below.

clear global Experiment;       % Delete any existing Experiment
TSwaitbar('clear');            % Kill any existing waitbars

TSinitexperiment('Manual_Example_100' , 100, [101 102], 'Mouse', 'King');   %Initialize the Experiment Structure

TSimporteventcodes('ManualCodes.txt');                              %Import the Event Code text file.

TSsetloadparameters('INPUTTIMEUNIT', 1, 'OUTPUTTIMEUNIT', 1, 'LOADFUNCTION', 'TSloadstdtab',
'FILEPREFIX', '!', 'FILEEXTENSION', ''); %Set our load parameters before loading

TSloadsessions( cd );                                           %Loads the data in the current directory

TSdeclareeventcodes;                                            %Declares our event codes globally

TSdefinetrial('ToneOrWNoise',[StartTrial1, EndTrial], [StartTrial2, EndTrial]); %Defines the first trial type, which
includes Tone Trials and White Noise trials

TSdefinetrial('ITI',[EndTrial StartTrial1], [EndTrial, StartTrial2]);      %Defines another trial type, which is the ITI
time in between trials.



%%% Analysis starts here

% First we are going to get a PokeCount for hole 1 and hole 2 when one of
% the trials is on. This is pretty straightforward, and we will not need
% any custom routines for this analysis.

TSsettrial('ToneOrWNoise'); % Set the current trial

TStrialstat('TriPokeCount', @TSparse, 'if (match == 1) result = [1 0]; else result = [0 1]; end', {[PokeOn1]
[PokeOn2]});
%Creates an N x 2 matrix containing 0's or 1's, with 1 in the first column
%for PokeOn1 events, and with 1 in the second column for PokeOn2 events.

TSapplystat('TriPokeSum','TriPokeCount', @sum, 1);
%Sums the previous matrix along hte rows, so that we end up with a 1 x 2
%array, with the first index being the number of Poke1's and the second
%being the number of Poke2's.
%
%Alternate method: Create an anonymous function which counts up the number
%of PokeOn1's and PokeOn2's.
%TSapplystat('TriPokeSum','TriPokeCount', ...
%@(tsdata)[sum(tsdata(:,2)==PokeOn1) sum(tsdata(:,2)==PokeOn2)]);
%
%This is just one other way, there are practically hundreds of ways to do
%this.


TScombineover('SesPokeCount', 'TriPokeSum');
%Combine the sums up from the trial level to the session level.

TSapplystat('SesPokeSum','SesPokeCount', @sum, 1);
%Sum the N x 2 poke count created by the combine over above, so that we get
%another set of 1 x 2 arrays.

TScombineover('SubPokeCount', 'SesPokeSum');
%Combine again up to the subject level
```

```
TSapplystat('SubPokeSum','SubPokeCount', @sum, 1);
%Sum again at the subject level, so that we have total poke count for hole
%1 and hole 2 for both subjects.


% Now we are going to do the same poke count analysis for the ITI's instead
% of the trials, so that we can compare how many more or fewer pokes the
% mouse produces during trials and during the ITI's.

TSsettrial('ITI'); % Set the current trial

TSsetdata('TSData');
%Make sure our data is set to TSData.

TStrialstat('TriPokeCountITI', @TSparse, 'if (match == 1) result = [1 0]; else result = [0 1]; end', {[PokeOn1]
[PokeOn2]});
%Now we are doing the same counting procedure, except for the ITI instead
%of the trials.

TSapplystat('TriPokeSumITI','TriPokeCountITI', @sum, 1);
% The 1 is to ensure that the sums are done along rows, this is a problem
% if there is only one match, cause then we will have a row vector, and
% summing that will produce a scalar.

TScombineover('SesPokeCountITI', 'TriPokeSumITI');
% Combine to Session level

TSapplystat('SesPokeSumITI','SesPokeCountITI', @sum, 1);
% Sum at session level

TScombineover('SubPokeCountITI', 'SesPokeSumITI');
% Combine at subject level

TSapplystat('SubPokeSumITI','SubPokeCountITI', @sum, 1);
% Sum at Subject level


% So far we have just been doing poke counts. Now we will do something a
% little different. Here we are going to look at the periods of time when
% one of the lights was on, and look at how long it took for the mouse to
% Poke that light. We will accumulate these times in a column vector. Then
% we will create 2 column data representing a histogram, using a helper
% function below.

TSdefinetrial('Light1On',[LightOn1, LightOff1]); %Define Light1On trial
TSdefinetrial('Light2On',[LightOn2, LightOff2]); %Define Light2On trial

TSsettrial('Light1On'); % Set Light1On to active trial

TStrialstat('TriLight1PokeTime', @TSparse, 'result = time(1) - starttime;', [PokeOn1 TSend]);
%Use TSparse to get first PokeOn1 after each LightOn1, and subtract its
%time from the LightOn1 time. These are built up in a column vector.

TScombineover('SesLight1PokeTimes', 'TriLight1PokeTime');
%Combine them up to the Session level as a larger column vector of poke
%times.
TScombineover('SubLight1PokeTimes', 'SesLight1PokeTimes');
%Combine again to get all poke times for that subject,

TSsettrial('Light2On'); % Set Light2On to active trial
```

```
TStrialstat('TriLight2PokeTime', @TSparse, 'result = time(1) - starttime;', [PokeOn2 TSend]);
%Use TSparse to do same analysis for PokeOn2.
TScombineover('SesLight2PokeTimes', 'TriLight2PokeTime');
%Combine them up to get another series of large column vector of poke
%times.
TScombineover('SubLight2PokeTimes', 'SesLight2PokeTimes');
%Combine again to get all poke times for this subject.

TSapplystat('SubLight1PokeHist', 'SubLight1PokeTimes', @MakeHistData);
TSapplystat('SubLight2PokeHist', 'SubLight2PokeTimes', @MakeHistData);
%Make histogram data for Light1 and Light2 for each subject. This can be
%plotted using the following TSapplystat call:
%
% TSapplystat('SubLight1PokeHist', @plotfunc);
% TSapplystat('SubLight2PokeHist', @plotfunc);
%
%Or it can be plotted using a button in TSExperimentBrowser.

TSapplystat('SubPokeTimes',{'SubLight1PokeTimes' 'SubLight2PokeTimes'}, @vertcat);
%Combine light1 and light2 poke times so that we have a collection of all
%poke times the subject made for light1 and light2. This is a measure of
%total responsiveness of the subject.

TSapplystat('SubPokeHist', 'SubPokeTimes', @MakeHistData);
%Make a histogram of the combined data. This can be plotted using
%the following TSapplystat call:
%
TSapplystat('SubPokeHist', @plotfunc);
%
%Or it can be plotted using a button in TSExperimentBrowser.

TSwaitbar('clear');
%Make sure the waitbars are cleared.

function [result] = MakeHistData (data) % This helper function converts poke times into histogram data
data = sort(data); %Sort the poke times
m = mean(data);    %Get the mean
stddev = std(data);%Get the standard deviation

xspace = linspace(m-1.5*stddev, m+1.5*stddev, numel(data) / 3); % All data within 1.5 std deviations of mean. Use numel(data)/3 as number of bins, this way there will be an average of 3 per bin, seems to be a nice number.
x = xspace(2:end) + xspace(1:end-1) / 2;          % x value's for plot are the average of bin cutoff values, so that the middle of the bins is the x value of the corresponding point.
y(numel(x)) = 0;                          % preallocate y array for speed

for n = 1:numel(data)                     % For each data point,
   idx = find(data(n) < xspace, 1);          % find the first bin greater than this data piece
   if idx > 1                         % if this not 1 or -1, so not off-end in either direction
      y(idx-1) = y(idx-1) + 1;             % increment that bin's y value
   end
end

result = [x' y'];                         % result is 2 column matrix with x in column 1 and y in column 2

function plotfunc(data) % Sample plotting function, very simple, plots histdata generated with above fcn.
figure;
bar(data(:,1),data(:,2));
```

The Experiment Browser

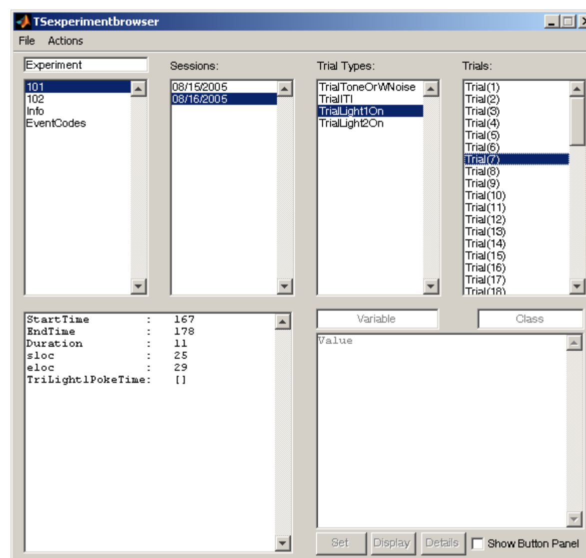
Introduction

TSexperimentbrowser;

This command will call up the TSexperimentbrowser gui. The Experiment Browser is a great tool for browsing through the Experiment structure and viewing the data it contains.

The browser works by allowing the user to select the fields and structures contained in Experiment from viewing panes. Subjects, and any other structures contained in the first level of the Experiment struct, appear in the first pane. Sessions, and any structures contained within the selected Subject, appear in the second pane. Trial definitions for the selected Session appear in the third pane, and the actual Trials themselves appear in the fourth pane. This allows you to select any substructure that exists within the Experiment.

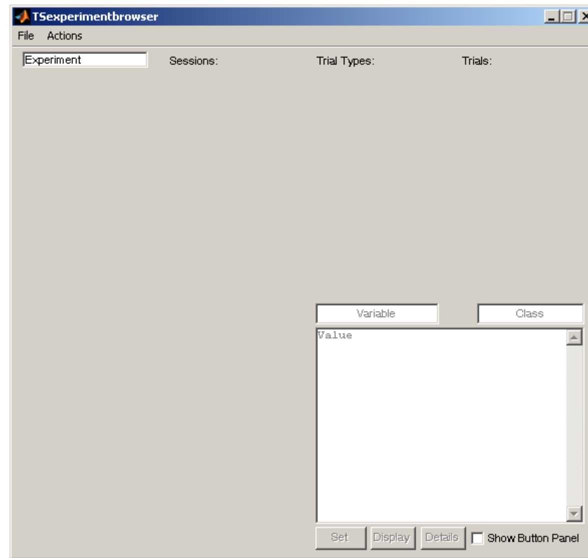
You should know that the Experiment browser uses the same copy of Experiment that MATLAB uses in the workspace; there are no local copies made for the browser. This means that you can modify the Experiment values in the MATLAB command line and the Experiment browser simultaneously without creating parallel copies of your data. The Experiment browser will update when you click on it again.



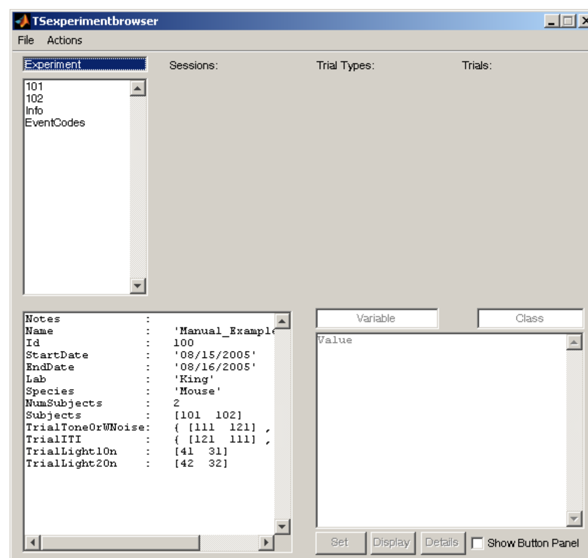
The four structure panes appear in the top half of the Gui. The bottom half of the gui is used for managing the fields of the current structure. In the bottom left, a large selection pane is used for displaying all the field names and values of the currently selected structure. In the bottom right, several text boxes and buttons can be used to manage and manipulate the current field.

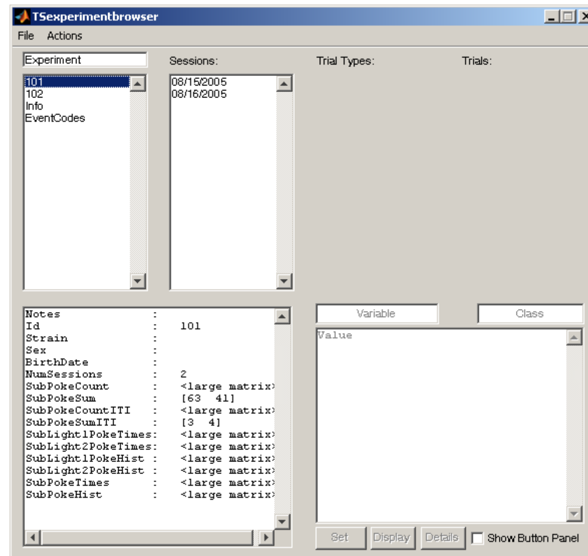
First, load the sample Experiment file into memory by executing the command:
TSloadexperiment('Manual_Example_100');

Now, call up the Experiment Browser by typing TSexperimentbrowser in the command line. (Alternatively, you can load browser first and then load the experiment from the "file" menu.) A window will popup, with the word Experiment appearing in the upper left and several buttons and panes appearing in the bottom right.

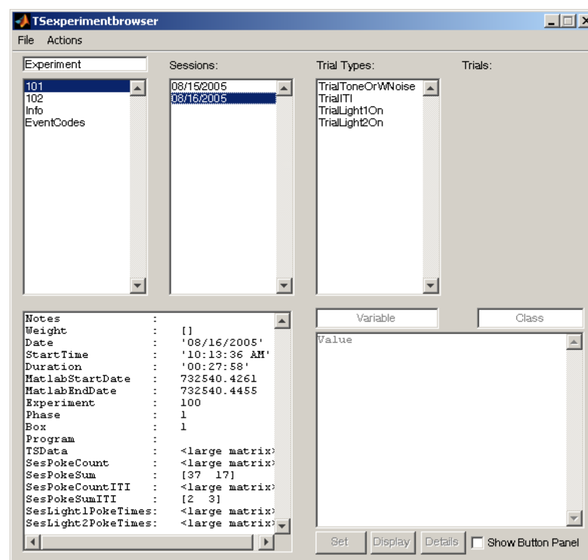


Start by clicking on Experiment. Because you have selected a structure, the base Experiment, two panes will now popup. The top one is the pane containing all the substructures in Experiment. The bottom one contains all the non-structure fields in Experiment. Select a subject from the top pane by clicking on one of the ID numbers. Now, another pane will appear to the right containing all the substructures of that subject.

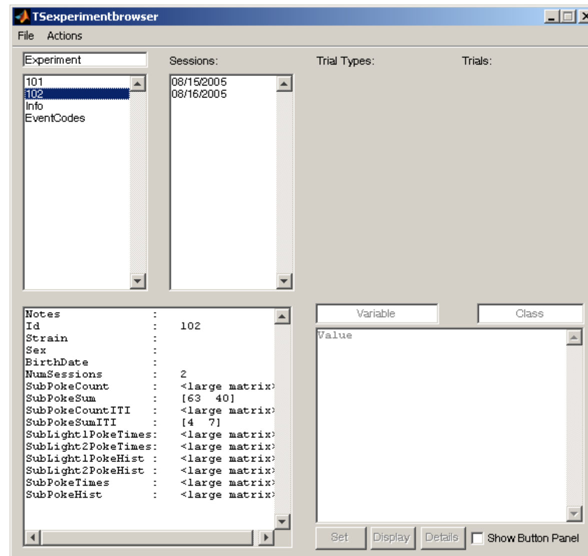




If you select one of the Sessions of that Subject in the pane that just appeared, any Trial definitions made for that Session and any substructures of it will appear in another pane to the right.

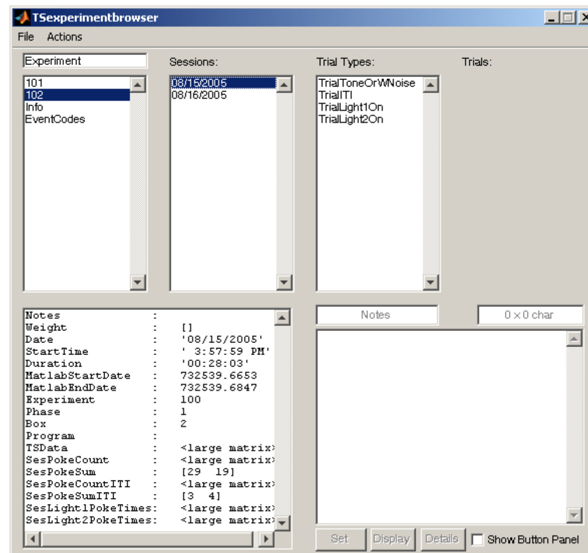


At any time, you can go back to a previous pane and select different items for browsing

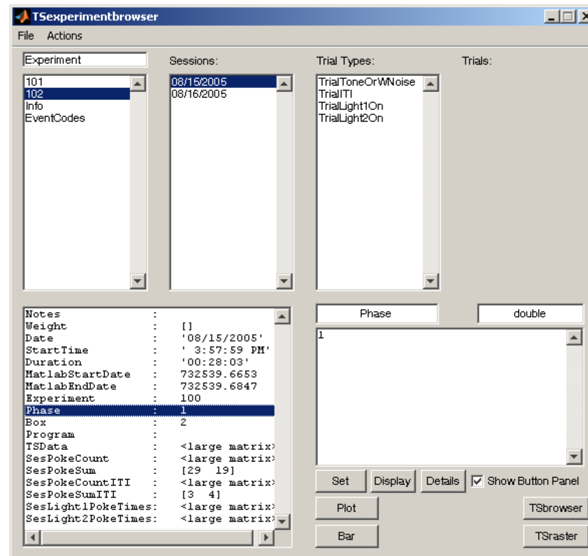


Browsing Fields

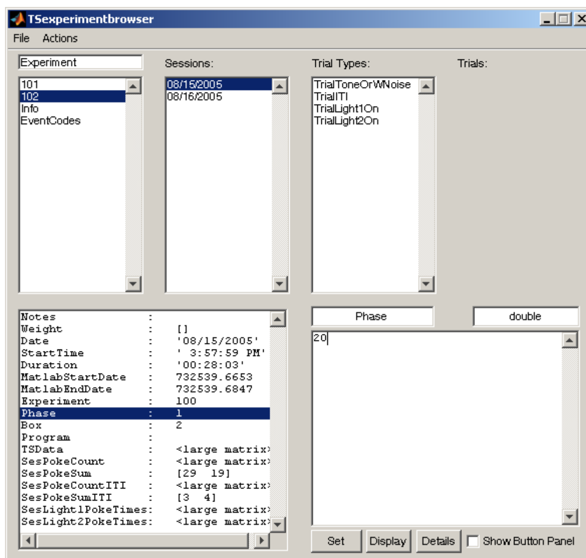
Now that you know how to browse through the structure tree, you will want to be able to select and view the fields of these structures. You have probably noticed the selection pane at the bottom left of the Gui. This pane always contains the fields of the most recently selected structure. For now, select a Session. You should see many fields appear in the Field pane below, with their names displayed to the left and their values displayed to the right:

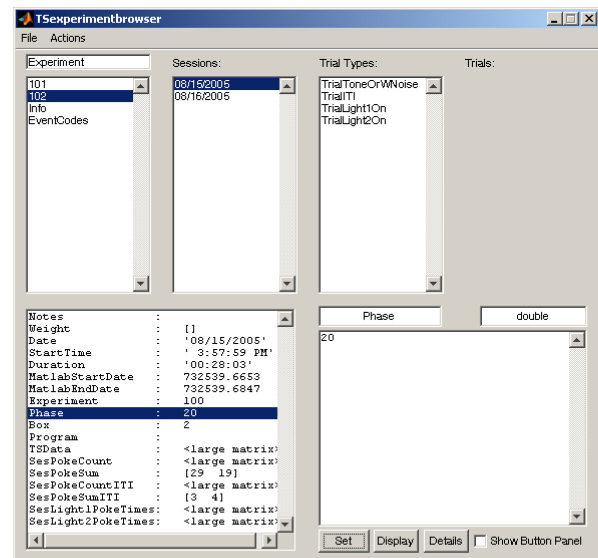


Some of these items include Notes, Weight, Date, Phase, Box, and TSData. For now, select Phase from the window.

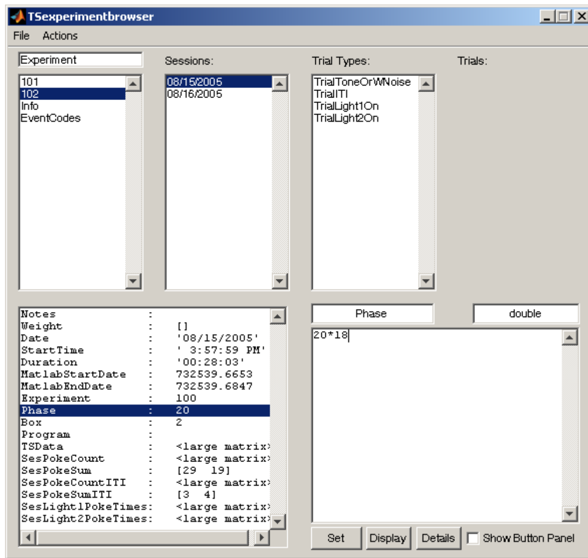


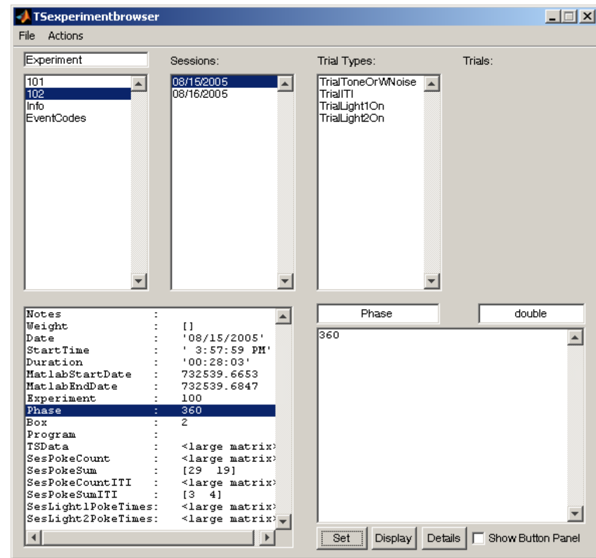
The boxes to the right will now become active because you have selected a field to manipulate. In the large textbox, you will see the value of Phase. You can edit this value by selecting in the box and changing it. When you are finished entering the new value, you can press the Set button to set it to the new value. The field pane at the left will be updated to confirm this change.



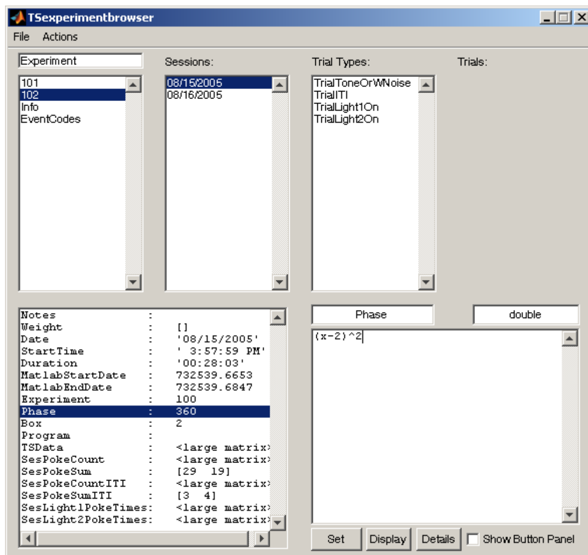


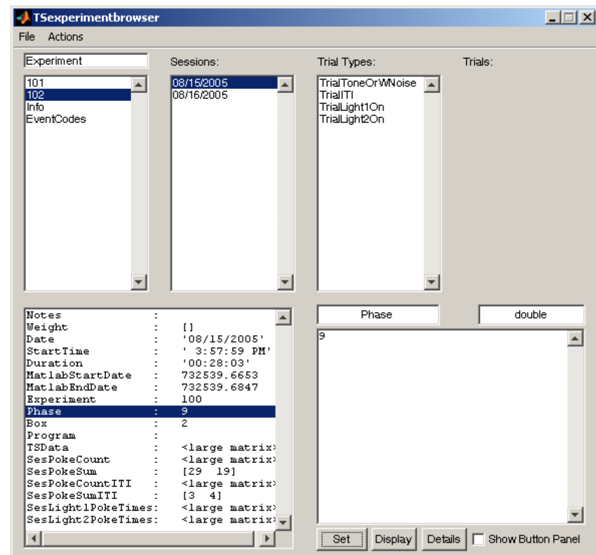
You can even type mathematical expressions in the text box, and they will be evaluated automatically when you press set. This is because the Browser uses the Eval function to evaluate the strings as MATLAB expressions, when the value is a numeric value. When you set them, they will be reduced to a single value.



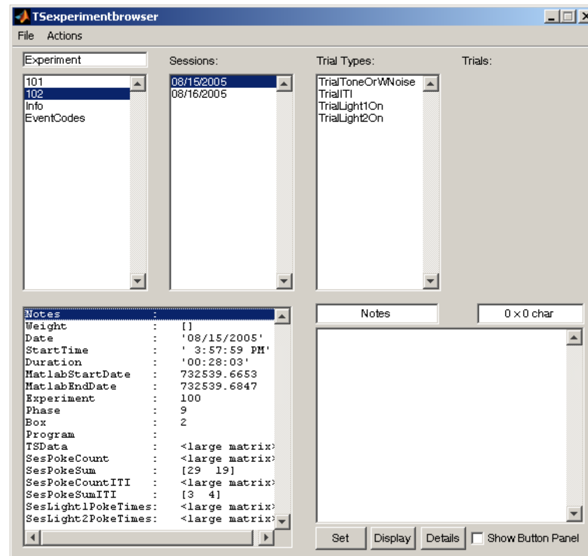


In fact, you can even use workspace variables in the window. For example, go to the MATLAB window and type x = 5 to set x as a workspace variable with value 5. You can now use x in an expression in the text box and MATLAB will evaluate it correctly:

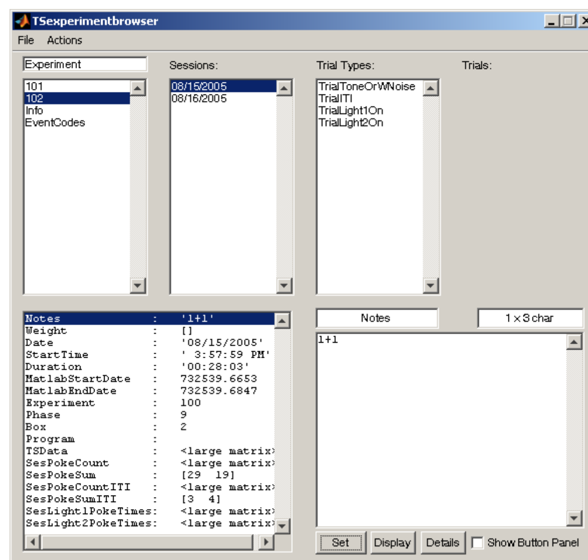




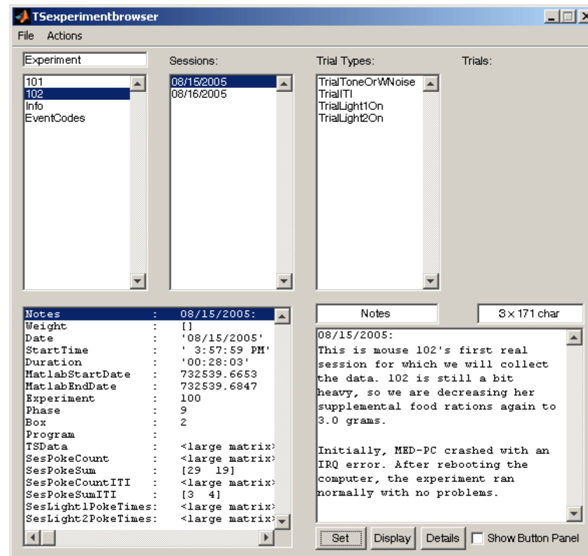
Now, try selecting a non-numeric field, such as Notes. Notes is of type char, not double, as you can see when you select it, in a box just above and to the right of the value textbox. At present, it is empty.



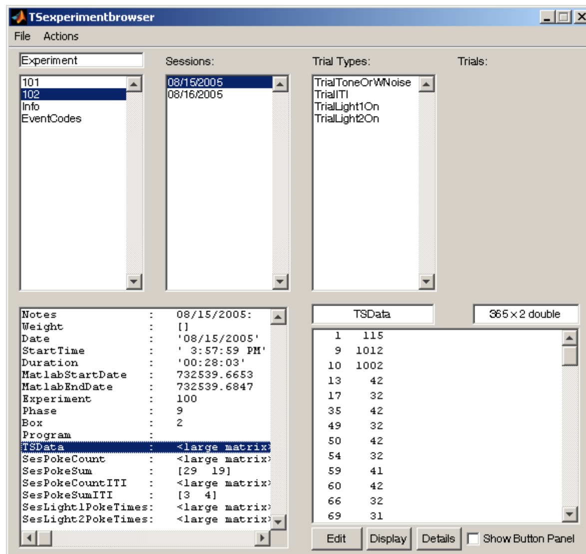
You can edit Notes by typing words in the Value box. Because Notes is a string, the Browser will not evaluate what you type, just save it in Notes. You can type 1+1 into the box and press set, and the value will be 1+1, not 2.

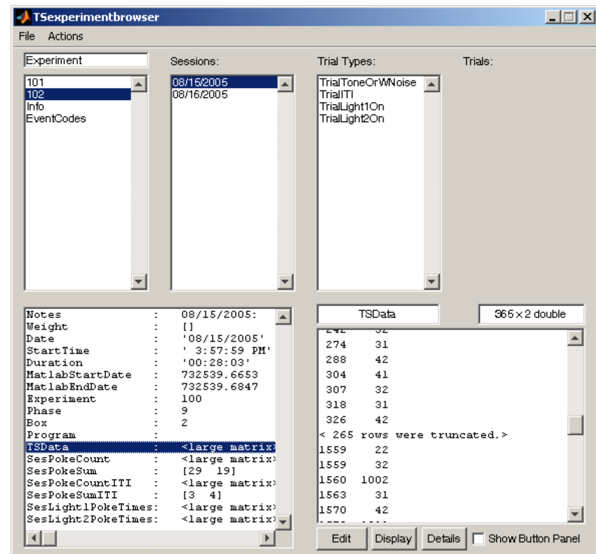


You can type as many lines of text as you wish; Notes will become a large char array:



Now try selecting a large numeric matrix, such as TSData. For matrices with multiple rows of data, the browser will display <large matrix> as the value in the field pane. It will try to display as many lines as it can in the textbox; however, we have put a limit of 100 lines of data displayed in the text box because MATLAB often hangs or freezes if you attempt to display more. If your matrix is more than 100 lines, the first 50 lines and the last 50 lines will be displayed, and in the middle of the textbox a note will inform you of how many lines were truncated from the display.





If you wish to view the whole matrix, you have two options. You can press the Edit button, which will open MATLAB's built-in array editor on the matrix. Or, you can press Display to simply dump the matrix to the MATLAB command window.

Loading and Saving with the Browser

Under the file menu, several features make it easier to manipulate Experiment files. You can load, save, create, or clear the Experiment structure from this menu.

Select Load Experiment from the file menu in order to load a new Experiment mat file. A file selector will popup, prompting you to select your experiment. If an Experiment already exists in the Workspace, you will be prompted before it is cleared.

Select Save or Save As from the file menu in order to save the current Experiment. A save window will popup if you select Save As, so you can pick where to put the file. If you press Save, the current MATLAB path will be used and the name of the Experiment will be the name of the file.

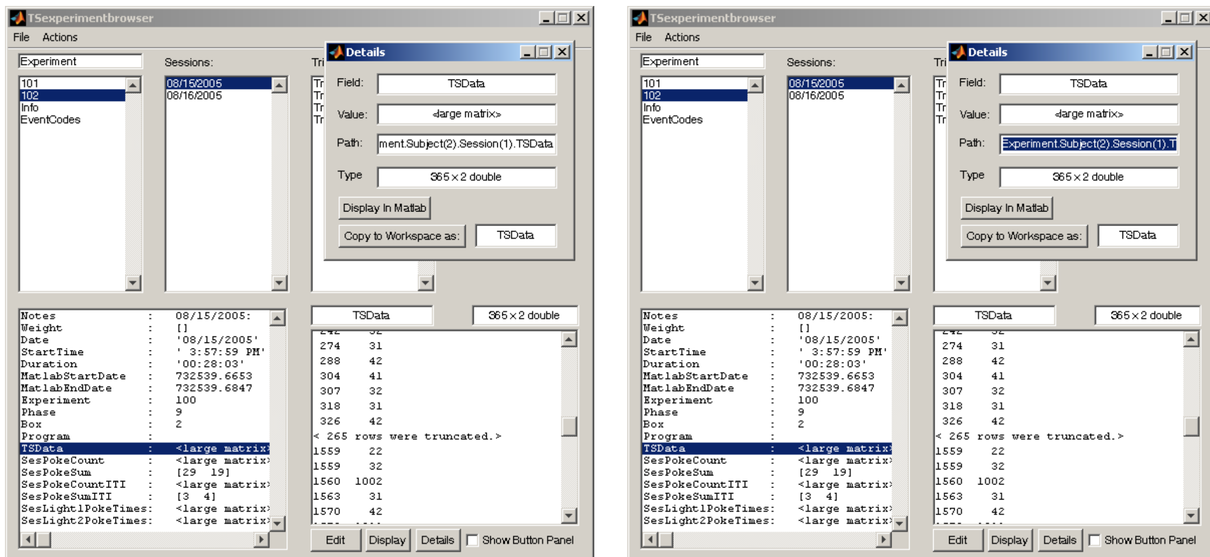
Select New Experiment from the file menu in order to create a new Experiment structure in memory. This option provides a nice gui-front for the TSinitexperiment function. You are prompted to give a name, number, the subjects, lab, and species of the Experiment. The last 2 are optional.

Select clear Experiment in order to clear the current Experiment, without saving it. This command is equivalent to the command clear Experiment entered in the command line.

Exit is not a feature in the file menu. Simply press the X in the corner to close your Experiment Browser window. Since the Experiment struct is in the MATLAB workspace, not the Browser, you don't have to worry about losing it unless you exit out of MATLAB as well.

Manipulating Fields with the Browser

Often you will want to do more than just look at your data. The browser can provide a great deal of information about your data with a few clicks and help you work much faster from the command line. You can click on the details button to bring up the Field Details window.



Details provides some of the same information from the previous window, but also it provides the "Path" and the "Copy to Workspace As" feature. Path is the full path from the Experiment base-struct to the selected Field, ex. Experiment.Subject(2).Session(1).TSdata as shown above. This is useful because you can use this in the command line, and you can copy-paste it out of the details window very quickly.

As a very similar alternative, you can press the copy to workspace as button in order to copy this value to another value in the workspace. By default, this new variable is the same name as your field. You can edit its name in the box next to the button. This is especially useful if you are very interested in that piece of data or wish to test out a function on some TSdata; you simply click the button and then use TSData as a variable in the command line.

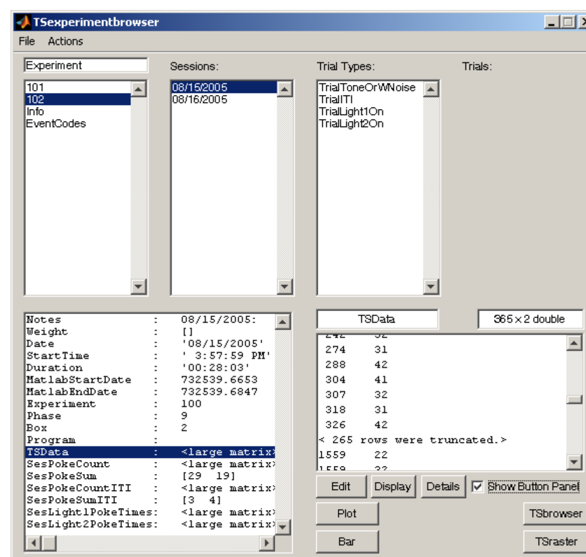
You may notice that all of the fields in the details window can be edited. This is simply a consequence of MATLAB not allowing us to make text fields that can be copy pasted but not edited. I thought it much more important that you are able to copy and paste key pieces of data quickly than it is that they might be inadvertently modified by the user. No changes made in the details window will cascade backwards to the actual value of the workspace or anything like that, even if you delete the fields and values and paths and etc.

You can close the display window safely by clicking the x.

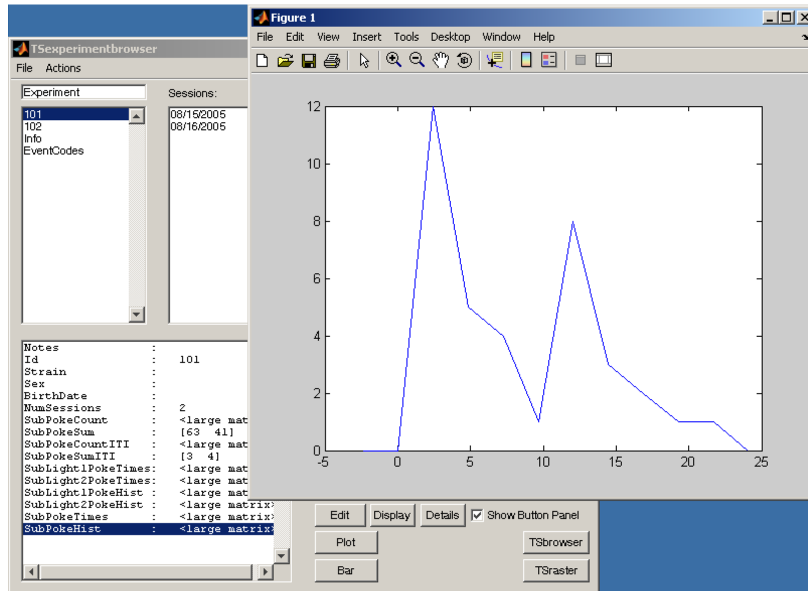
Advanced Features

Another powerful feature of the browser is the Button Panel. The button panel is a set of customizable buttons for manipulating fields and executing functions and code blocks on data. 8 button slots are available.

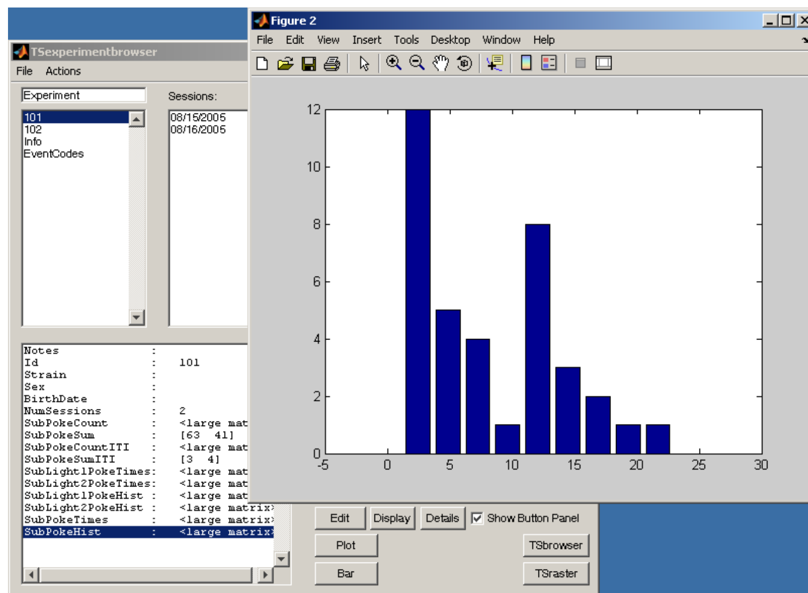
To display the button panel, click on the Show Button Panel checkbox.



In the above screen shot, only 4 of the 8 buttons are visible. This is because only 4 of them are active. Inactive buttons are invisible. The first button is labeled Plot. Press it to quickly plot the selected data. This plot is not a great example. Going back to Subject 101, SubjectPokeHist is a much better plot example.



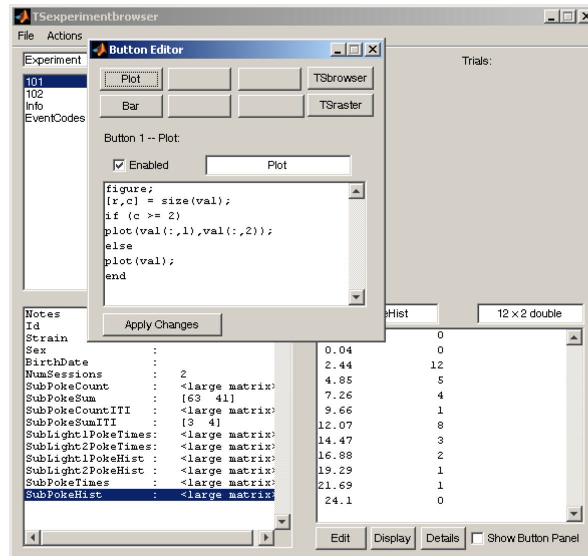
Since that data is actually histogram data, however, it makes more sense to use the bar function instead, via the bar button



The other 2 functions, are TSbrowser and TSraster. TSbrowser is a feature that is in development and not yet released. TSraster will be explained in greater detail below.

Modifying the Custom Buttons

You can modify the custom button set by selecting Edit Buttons from the Actions menu. The Button Editor will then popup:



At the top, all the button slots are displayed in the same configuration as they are in the panel. To modify a button, click on it. At the bottom, a checkbox will show whether it is enabled, what its name is, and what its corresponding MATLAB code is. You can modify these, and then press the Apply Changes button to apply the changes. Then, you can select another button, and press the X when finished.

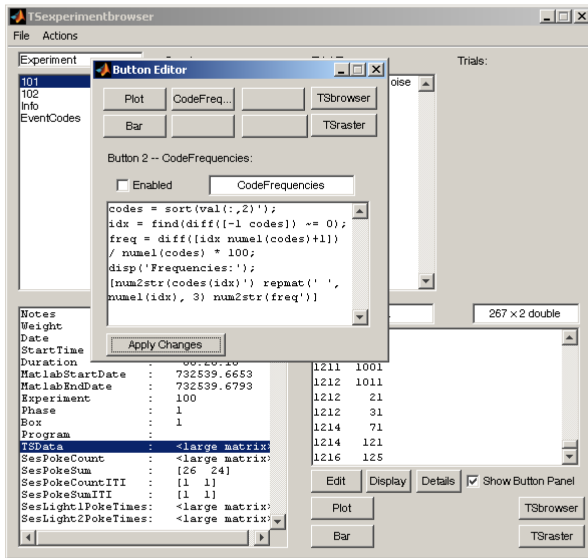
As you can see above, the code block for the plot function makes use of the variable val. Val, in the context of button code, is the currently selected value. In this case, it would be the large matrix the user wishes to plot.

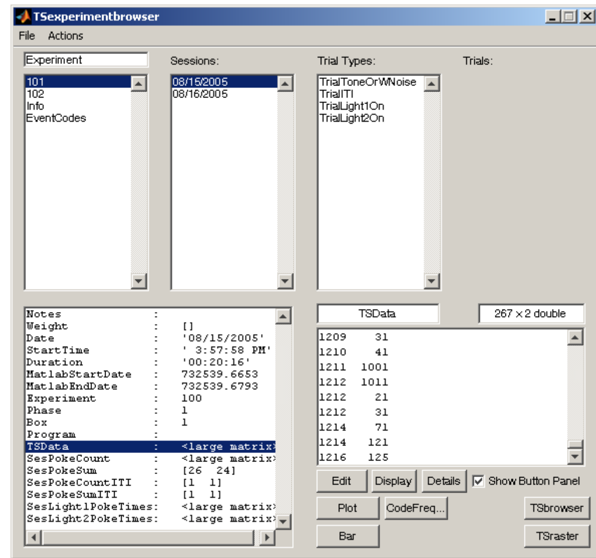
Three “special variables” are provided to blocks of button code at run time: var, val and path. Var is the name of the currently selected variable. Val is the value of the currently selected variable. Path is the path to the currently selected variable. All three are the same as those displayed in the details box.

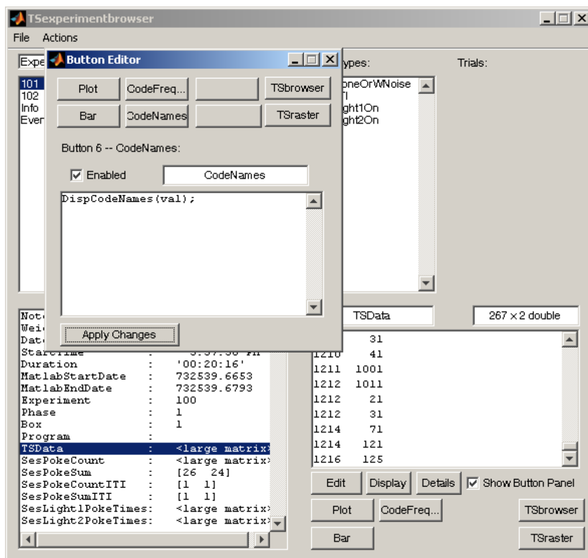
Also in the scope of the button code is the Experiment structure. You can see an example of this being used in the TSraster code block, because the function it calls requires the currently Active Trial as an input.

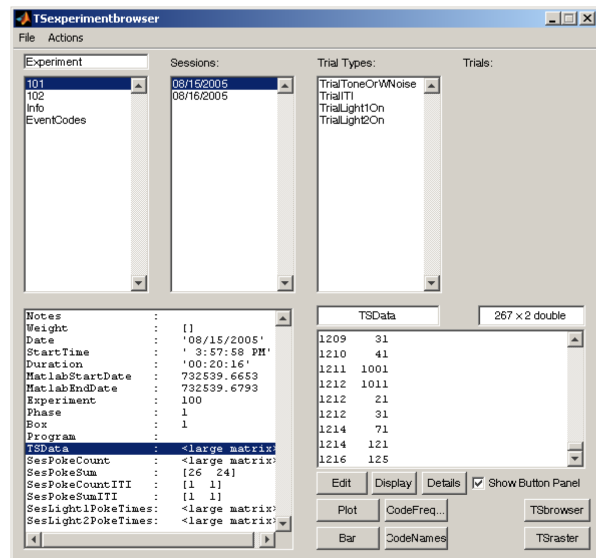
Changing the value of val in the button code does not cascade back to actual value in the Experiment structure and cause it to be changed. There are 2 reasons for this. First, it makes the system more flexible in terms of what code you can write without destroying the data. Second, we encourage users to execute functions using the ApplyStat function and similar methods so that they can replicate exactly what they did and create an M-File which will generate their analysis directly from the data. Clicking the button leaves no such record and makes it difficult to retrace steps and track down bugs and errors. Also, apply stat is generally faster since it operates on all the fields at once, whereas buttons only manipulate one field at a time. However, if you still want to modify fields from within the button code, the easiest way to do so is using the eval function and the path variable. Since the Experiment structure is within scope of the code, you can write eval([path ‘ = ‘ num2str(somevalue) ‘;’]); to modify the current value.

You can typically create buttons for your own functions quickly and without a lot of hassle using the special variables and the Experiment structure.









If your function requires more information than is available from the special variables and the Experiment structure, I would recommend using one of the UI dialogs built into MATLAB to get more information from the user, such as inputdlg, questdlg, and listdlg. This is especially nice because it helps to integrate the button with the rest of the gui, and it is very natural from the user's perspective for a button press to launch a gui dialog. Alternatively, you can write your own modal dialog box using GUIDE, and call that gui instead.

Currently, the experiment browser automatically saves your button settings in the matlab preferences. (You can read more about this in the documentation for "getpref", "ispref", and "setpref".) These preferences are specific to each computer. If you run the browser on different computers they will have different button sets. If you want to move the button settings around, you can go to file => import button settings or file => export button settings, which will import or export a collection of buttons from a .mat file. You can also restore factory default buttons from the "actions" menu. If you want to send your experiment to a colleague or post it on the internet, all you need to do is give them the Experiment, the buttons settings file, and any other functions or settings files they will need and they can view your data with the same custom button set that you have.

The best uses of the buttons are for calling up specialized plots and graphs of data and for calling client functions from the browser. Buttons can make it much easier to look at the day's data or the product of a statistical analysis in

some field of the Experiment structure using all the power of MATLAB's graphing tools and functions and putting them only one click away. Also, buttons can integrate many user-defined functions into an easy to use interface, increasing their value and usefulness.

As you can see, the Experiment Browser is a very powerful tool that can make it much more efficient and convenient to use the Experiment structure. From visually browsing the whole structure with a few clicks, to copying paths and values out of the details window, to creating flexible and powerful buttons to perform routines quickly, the Experiment Browser can really enhance and improve the way you use the TS library to process and analyze your data.

Raster Plots

TSraster

Introduction

One of the most general and useful visualizations of time-stamped data is a raster plot. A raster plot plots events as points on a 1 dimensional time axis. Since all time stamped data consists of times and events, this means that any TSdata can be plotted this way.

The TS library function to create raster plots is TSraster.

TSraster(tsdata, trialdef, events, arg1, arg2, arg3, arg4)

TSraster has 3 mandatory arguments: the TSdata to be used, the trial definition to use to break up the TSdata, and the events to plot. You should be familiar with TSdata, and trialdef is simply any legal matchcode used with TSmatch or TSdefinetrial. Events is a 2 column matrix of event codes, in which each row of the matrix defines an event. An example of an event might be when a feeding event occurs. In this case, there might be a code, Feed1, which indicates that feeder 1 dispensed a pellet at a certain time. This is a point event, an event which is thought to happen at a single point in time, and thus is plotted as a point in a raster plot. A duration event might be when a light comes on. For example, there might be 2 codes, LightOn1 and LightOff1. This pair of codes constitutes a duration of time, and thus would be plotted as a series of broken horizontal lines in a raster plot, indicating that the light was on at the points where the line is present and off where it is not.

Events is a 2 column matrix, so in the case of duration events, the On code goes in the 1st column, and the Off code goes in the 2nd column. For point events, place the point code in the 1st column and a 0 in the 2nd column.

The optional arguments allow you to specify the Colors, Offsets, and Labels for these events. If they are not given, default values will be chosen. They can be given an any order and included or excluded in any combination.

Colors should be a string the same size as the number of events. Each character specifies the color or marker used for the corresponding event. The options for these characters can be viewed if you type help plot in the command window. If you would like to specify more than one code for some of the events, then you should pass a character matrix with the same number of rows as events, and then each row of characters will correspond to an event. This matrix can be easily constructed using the strvcat function. For example, if you wanted the first event to be a red line, the second to be a blue square, and the third to be a green dotted line, your character matrix might be: strvact('r' 'bs' 'g:')

Offsets should be a vector of integers the same size as the number of events. This vector allows you to offset the horizontal lines on which the events are plotted so that they are not all on top of each other. The integer does not correspond to an exact number of pixels; the integer offsets are all evenly spaced so that the distance from the

largest offset number to the smallest is the same in every raster plot. This allows you to make however many offset lines you need. By default every event is placed on a separate line.

Labels is a Cell String in which each string corresponds to an event. If this argument is present a legend will be added to the plot using these labels.

Lastly you can pass a handle to an Axes object, and then this axes will be cleared and the raster will be plotted on it. You probably will not need to use this.

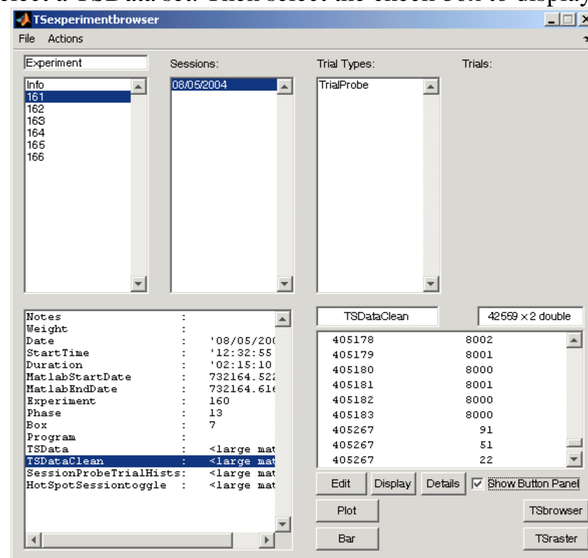
As you can see TSraster is a very powerful tool, but it can be tedious to type all the information.

The Raster Gui is a graphical tool designed to facilitate the creation of raster plots. It acts as a graphical interface to the TSraster function. When using the TSrastergui, you can select Trial definitions from a drop down menu, or enter your own, you can select event codes from drop down menus, or enter your own, you can select colors, markers, and linestyles from a drop down menu, and you can add labels. You can add these event definitions to the current raster group, delete them, or modify them. You can also maintain multiple raster groups at once, and you can load or save raster groups to .mat files for backups or sharing with others.

Loading the Raster Gui

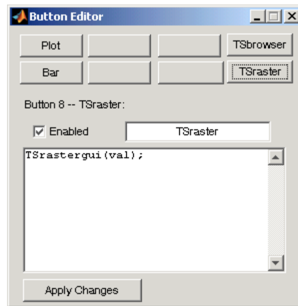
Tsrastergui(tsdata);

This command will call up the TSrastergui. The Raster GUI takes tsdata as an initial argument, and provides a graphical user interface for generating raster plots depicting events as they occurred during trials for any given session. The easiest way to call the Raster GUI is through the Experiment Browser. Start the Experiment Browser (TSexperimentbrowser) and select a TSData set. Then select the check box to display the button panel.

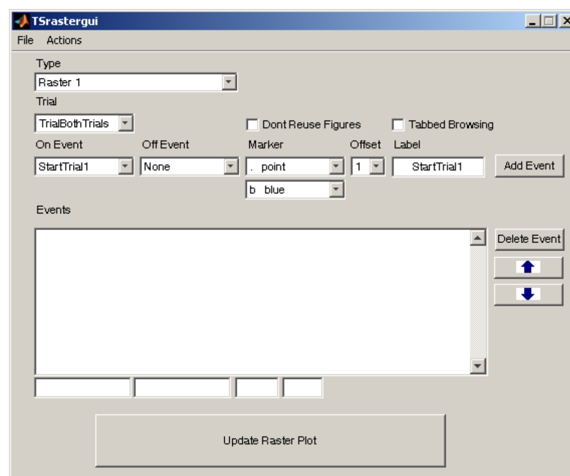


For further help and instructions refer to the Experiment GUI documentation.

Once you have selected a TS Data set you can simply click on the TSraster button to display the Raster GUI. If the TSraster button is not currently in your button panel you can add it by using the “Action” menu and selecting “Edit Buttons” (or by using the keyboard shortcut Ctrl-B).

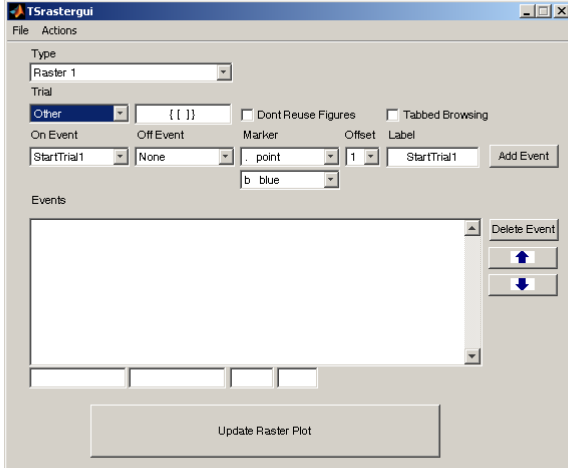


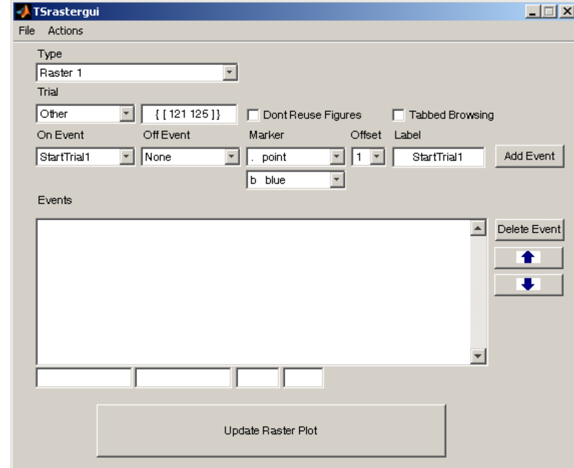
Using the button editor, create a new button with the value "Tsrastergui(val);" and click the "Apply Changes" button. This will ensure that the Raster GUI will be passed the selected TS Data when it is called. Then go to the Session data or other TSdata that you would like to raster, and press your TSrastergui button. After you have started the Raster GUI you will be presented with the following interface:



At the upper left you will see a drop down menu that indicates the current raster you have selected. It should say "Raster 1" at the start, (unless someone else has been using TSrastergui before you. If they have, go to file and select "New Raster Settings"). This drop down menu keeps track of all of the raster settings you currently have loaded. If you have more than one, you can switch between them using this menu.

Directly beneath it, you will see another drop down menu, marked Trial. This list is populated with the names of trials that you have already defined using TSdefinetrial. You can select them here and you won't have to retype them. If you want to make up a trial definition now to use, you can do that by selecting other, and typing the trial definition in the field that appears to the right.



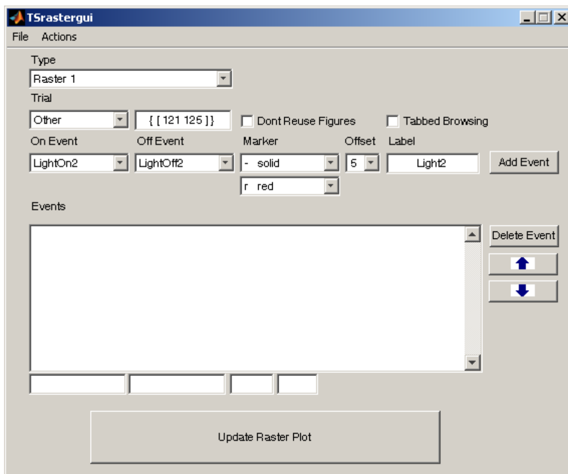


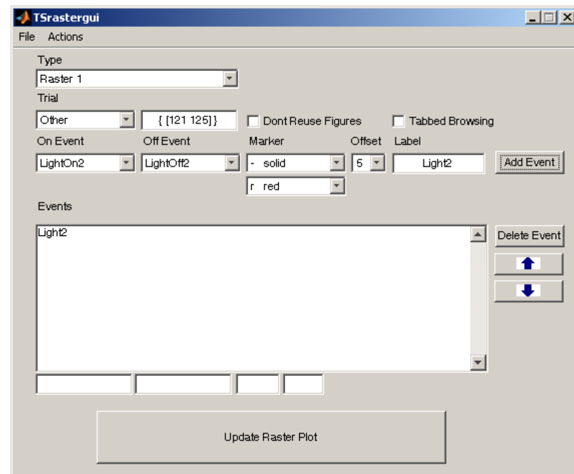
Now that you have a trial definition for your data, you need to start defining events. The rest of the drop down menus are all for that purpose. First you set them to the appropriate settings, then you select "add event" to add this new event to your list is of event definitions.

The first 2 drop down menus are marked On event and Off event. They are populated with your event code definitions for this Experiment. They also can be marked other, in which case a field will appear where you can enter the number code you wish to use. The Off event can be marked None, which will indicate a point event.

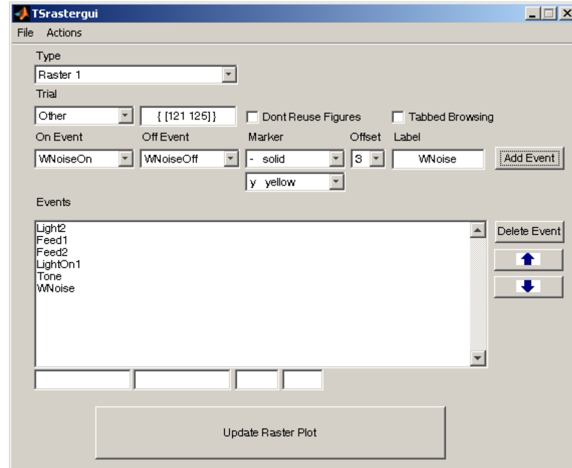
You can also select your colors and marker very easily. If you have selected a point event, then the marker drop downs will allow you to choose a color and a point marker. If you have selected a duration event, then you can select a color and a line style. Be warned, it may be unadvisable to use a dotted line or dashed line marker if the event you are plotting tends to go on and off frequently, as you will be unable to tell the dashes from when the event actually went on or off.

You can select the offset number from the offset dropdown. You can select a number between 1 and 10. Finally, you can select your label. It defaults to the name of the On event when you set it, but you can type whatever you like.

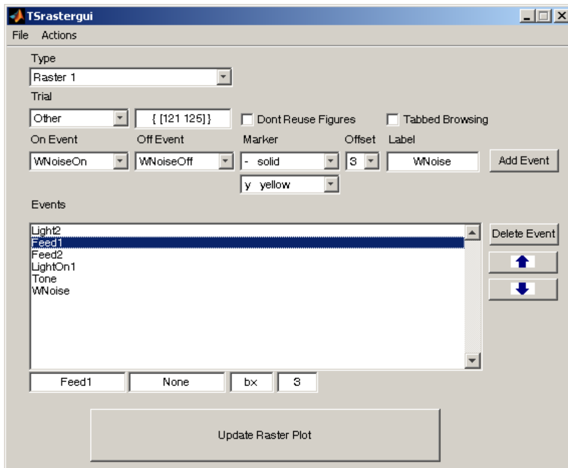


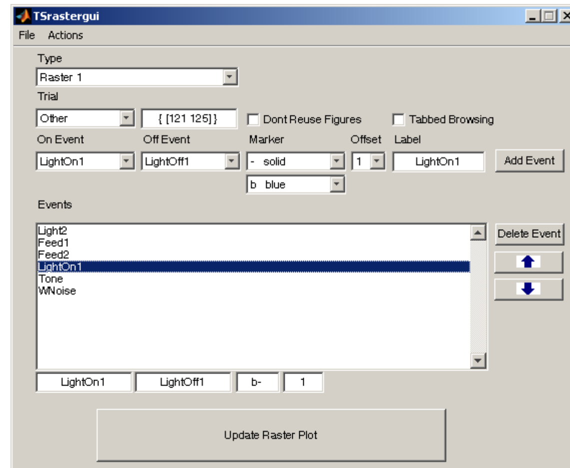


Now you can add your event. Click Add Event, and its label will appear in the "events" list. Add a few more events using different codes and markers.

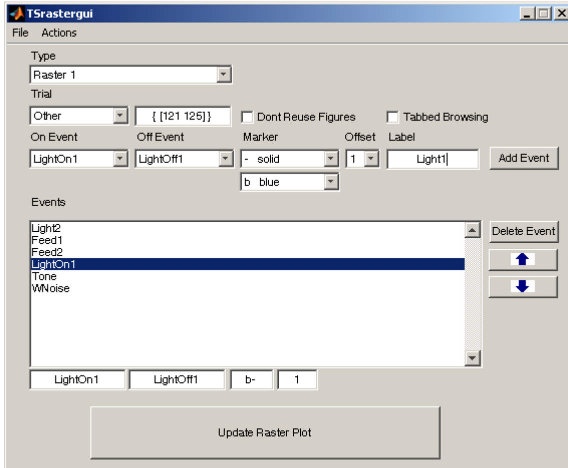


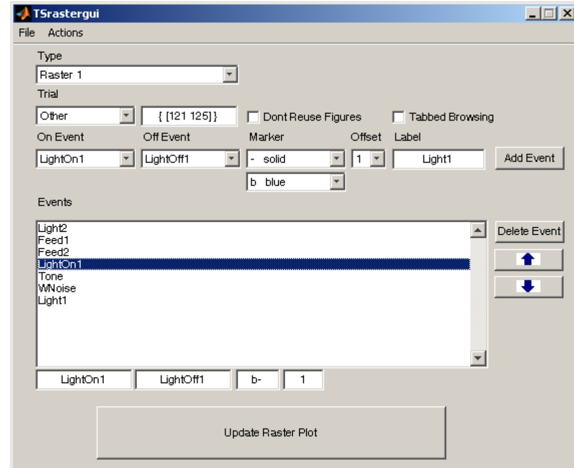
Now you have the makings of a good raster plot. To review your settings, you can select one of the events, and its configuration will appear in the text boxes below the events box. You can quickly browse through your current events this way.

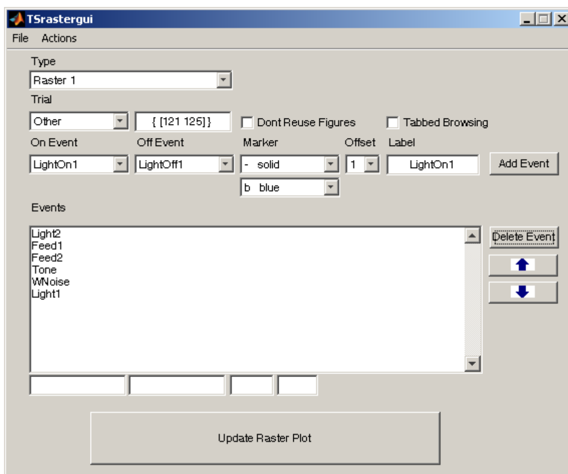


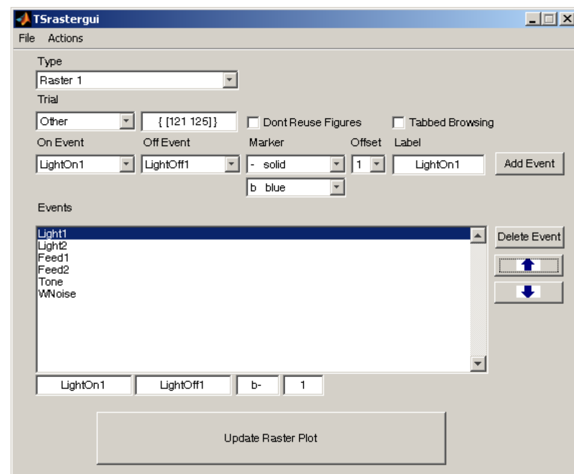


If you double click on an event, then the popup menus for On/Off events, Markers, Offsets, and the Labels textbox will all be reset to reflect that event's current state. If you need to add several similar events, you can use this to quickly select the right settings. This is also very useful if you discover an error in one of your events. You can double click it, fix the error and add the changed version, and then select the flawed event and press delete. You can use the arrows to move around one or more of the events into order.

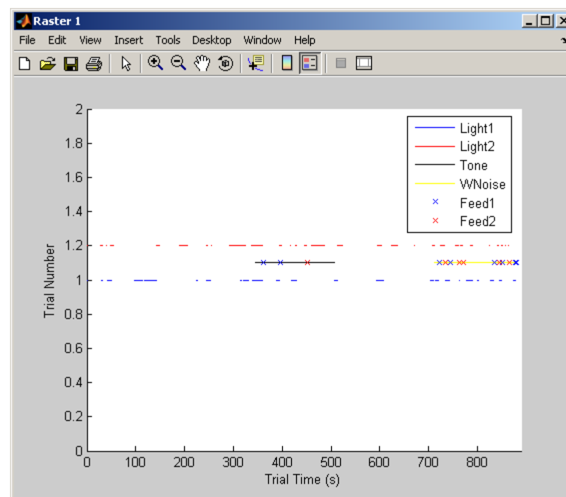








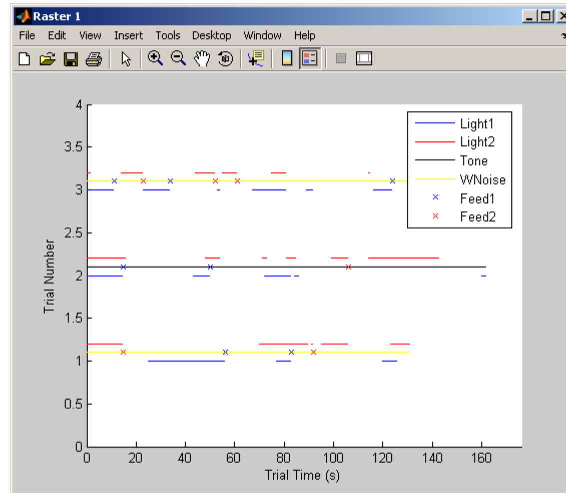
Now that your events are all correct, you are ready to make your plot. Press the large button at the bottom to make your plot appear.



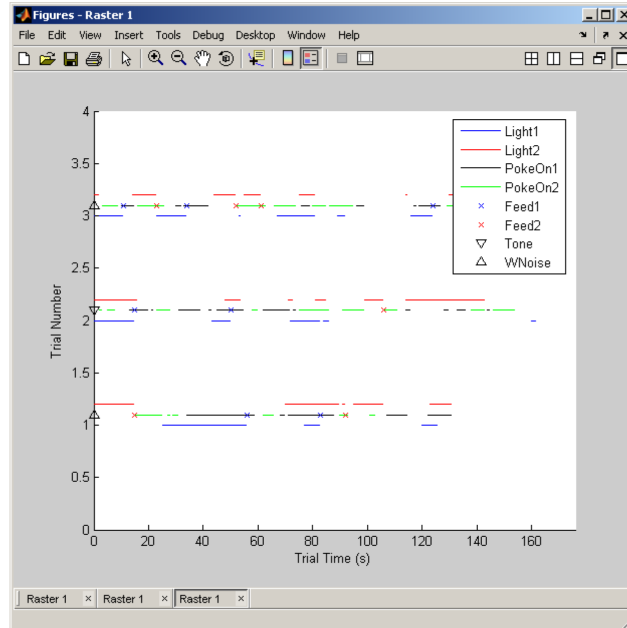
Notice the 2 checkboxes at the top, labeled “Don’t Reuse Figures” and “Tabbed Browsing”. These checkboxes change what happens when you press that button. The reuse figures option lets you select whether you want a new figure window made every time you press the button or if you want to reuse the windows for each type of raster.

Reusing can be quite handy when you are setting up your events; you can press the update button every time you add an event to see if it turned out the way you thought it would, then you can quickly tweak it and update again to see what effect your changes had.

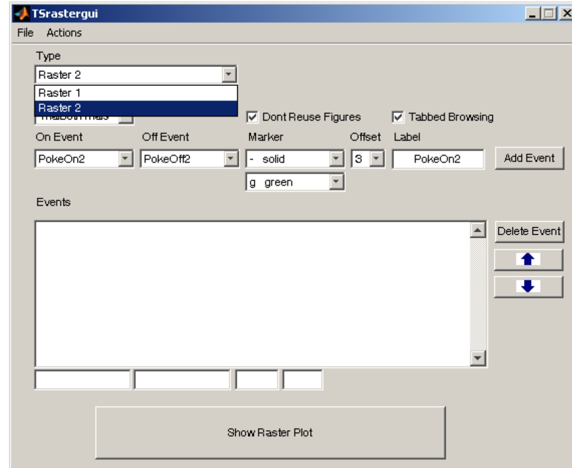
In this example, the Trial selection was changed to produce a different (and more interesting) breakdown of the session. With 2 clicks this change can be instantly updated to the same figure window we used above to reflect the change.



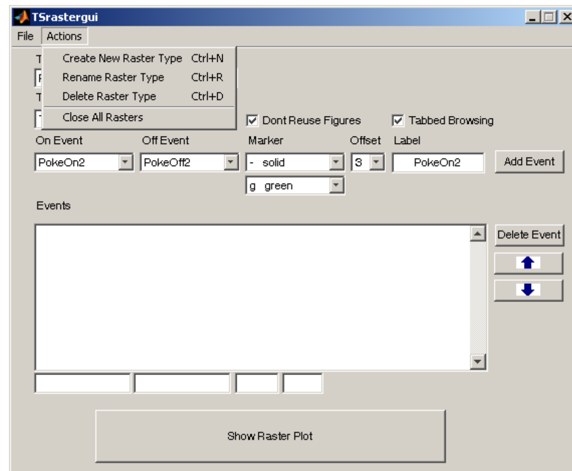
The tabbed browsing check starts the figure windows in “Docked” mode. Since you can quickly accumulate many figure windows when making raster plots, it can be handy to check this and keep them all organized.



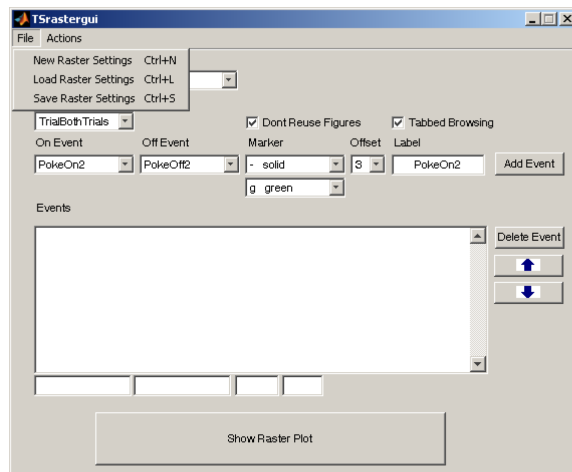
To make a second raster definition, go to the “Actions” menu, and select “Create New Raster Type” You will be asked to give the name of the new type, and then a blank event list will be shown. You can now switch back and forth between the 2 rasters using the upper left popup menu. Events will be added to the currently selected raster. You can make plots of either, and even if you have set it to reuse figure windows, raster types will not overwrite each other’s figures.



Also in the actions menu, you can rename or delete a raster type group. If you have accumulated too many figures, you can quickly delete all the raster figures using the “Close all rasters” button.

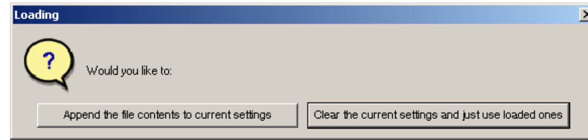


If you would like to save a collection of raster types, you can save it from the file menu.



You can also load a collection, and then you must select whether to append the current settings with the file contents or to overwrite the current settings and just use the file contents. So, if you just opened the raster gui and don’t have

any important raster types made, you can just clear it. If you just made a new one and you also want 2 old ones you made, then you should append.



If you have raster plot settings that are particularly useful for interpreting your data, you should distribute these settings files with your Experiment.mat, and then other TSlib users can make the rasters and examine your data on their own.

All the first session data

Row #:	Time:	Event:	Codename:
1:	1	115	StartSession
2:	80	42	LightOn2
3:	87	32	LightOff2
4:	130	42	LightOn2
5:	132	32	LightOff2
6:	151	41	LightOn1
7:	168	42	LightOn2
8:	169	32	LightOff2
9:	174	42	LightOn2
10:	175	31	LightOff1
11:	185	32	LightOff2
12:	188	42	LightOn2
13:	201	112	StartTrial2
14:	201	81	WNoiseOn
15:	216	1012	PokeOn2
16:	216	22	Feed2
17:	216	32	LightOff2
18:	226	1002	PokeOff2
19:	226	41	LightOn1
20:	228	1012	PokeOn2
21:	229	1002	PokeOff2
22:	230	1012	PokeOn2
23:	232	1002	PokeOff2
24:	235	1011	PokeOn1
25:	257	21	Feed1
26:	257	31	LightOff1
27:	260	1001	PokeOff1
28:	263	1012	PokeOn2
29:	267	1002	PokeOff2
30:	269	1011	PokeOn1
31:	271	1001	PokeOff1
32:	271	42	LightOn2
33:	272	1011	PokeOn1
34:	278	41	LightOn1
35:	284	21	Feed1
36:	284	31	LightOff1
37:	289	1001	PokeOff1

38:	291	1012	PokeOn2
39:	291	32	LightOff2
40:	292	42	LightOn2
41:	293	22	Feed2
42:	293	32	LightOff2
43:	294	1002	PokeOff2
44:	296	42	LightOn2
45:	302	1012	PokeOn2
46:	304	1002	PokeOff2
47:	307	32	LightOff2
48:	308	1011	PokeOn1
49:	316	1001	PokeOff1
50:	321	41	LightOn1
51:	323	1011	PokeOn1
52:	324	42	LightOn2
53:	327	31	LightOff1
54:	332	71	WNoiseOff
55:	332	121	EndTrial
56:	332	41	LightOn1
57:	334	32	LightOff2
58:	336	31	LightOff1
59:	338	1001	PokeOff1
60:	360	42	LightOn2
61:	362	41	LightOn1
62:	365	31	LightOff1
63:	366	32	LightOff2
64:	372	42	LightOn2
65:	374	41	LightOn1
66:	375	32	LightOff2
67:	381	42	LightOn2
68:	385	31	LightOff1
69:	389	32	LightOff2
70:	429	41	LightOn1
71:	434	1011	PokeOn1
72:	435	1001	PokeOff1
73:	448	31	LightOff1
74:	449	41	LightOn1
75:	474	42	LightOn2
76:	477	31	LightOff1
77:	479	1012	PokeOn2
78:	481	1002	PokeOff2
79:	483	32	LightOff2
80:	522	42	LightOn2
81:	540	32	LightOff2
82:	556	41	LightOn1
83:	561	31	LightOff1
84:	576	41	LightOn1
85:	576	42	LightOn2
86:	581	32	LightOff2
87:	586	42	LightOn2
88:	587	31	LightOff1
89:	590	32	LightOff2
90:	623	42	LightOn2
91:	646	41	LightOn1
92:	650	31	LightOff1
93:	652	41	LightOn1

94:	658	32	LightOff2
95:	664	31	LightOff1
96:	669	42	LightOn2
97:	671	41	LightOn1
98:	678	111	StartTrial1
99:	678	61	ToneOn
100:	680	1012	PokeOn2
101:	681	1002	PokeOff2
102:	683	1012	PokeOn2
103:	686	1002	PokeOff2
104:	691	1011	PokeOn1
105:	693	21	Feed1
106:	693	31	LightOff1
107:	694	32	LightOff2
108:	698	1001	PokeOff1
109:	699	1011	PokeOn1
110:	700	1001	PokeOff1
111:	701	1012	PokeOn2
112:	706	1002	PokeOff2
113:	709	1011	PokeOn1
114:	717	1001	PokeOff1
115:	720	1011	PokeOn1
116:	721	1001	PokeOff1
117:	721	41	LightOn1
118:	723	1011	PokeOn1
119:	726	42	LightOn2
120:	728	21	Feed1
121:	728	31	LightOff1
122:	732	32	LightOff2
123:	733	1001	PokeOff1
124:	736	1012	PokeOn2
125:	738	1002	PokeOff2
126:	740	1011	PokeOn1
127:	749	42	LightOn2
128:	750	1001	PokeOff1
129:	750	41	LightOn1
130:	751	1011	PokeOn1
131:	751	32	LightOff2
132:	752	1001	PokeOff1
133:	753	1012	PokeOn2
134:	759	42	LightOn2
135:	761	31	LightOff1
136:	762	41	LightOn1
137:	763	32	LightOff2
138:	764	1002	PokeOff2
139:	764	31	LightOff1
140:	769	1012	PokeOn2
141:	777	1002	PokeOff2
142:	777	42	LightOn2
143:	784	1012	PokeOn2
144:	784	22	Feed2
145:	784	32	LightOff2
146:	789	1002	PokeOff2
147:	792	1011	PokeOn1
148:	792	42	LightOn2
149:	794	1001	PokeOff1

150:	806	1011	PokeOn1
151:	808	1001	PokeOff1
152:	811	1011	PokeOn1
153:	814	1001	PokeOff1
154:	816	1012	PokeOn2
155:	821	1002	PokeOff2
156:	821	32	LightOff2
157:	822	1011	PokeOn1
158:	823	1001	PokeOff1
159:	824	1012	PokeOn2
160:	832	1002	PokeOff2
161:	838	41	LightOn1
162:	840	51	ToneOff
163:	840	121	EndTrial
164:	846	42	LightOn2
165:	848	31	LightOff1
166:	858	32	LightOff2
167:	925	41	LightOn1
168:	927	42	LightOn2
169:	938	32	LightOff2
170:	942	31	LightOff1
171:	956	42	LightOn2
172:	971	32	LightOff2
173:	1002	42	LightOn2
174:	1004	32	LightOff2
175:	1036	41	LightOn1
176:	1040	42	LightOn2
177:	1043	112	StartTrial2
178:	1043	81	WNoiseOn
179:	1043	31	LightOff1
180:	1045	41	LightOn1
181:	1045	32	LightOff2
182:	1046	1012	PokeOn2
183:	1052	1002	PokeOff2
184:	1053	1011	PokeOn1
185:	1054	21	Feed1
186:	1054	31	LightOff1
187:	1057	42	LightOn2
188:	1058	1001	PokeOff1
189:	1059	1012	PokeOn2
190:	1066	22	Feed2
191:	1066	32	LightOff2
192:	1066	41	LightOn1
193:	1069	1002	PokeOff2
194:	1073	1011	PokeOn1
195:	1075	1001	PokeOff1
196:	1076	1011	PokeOn1
197:	1077	21	Feed1
198:	1077	31	LightOff1
199:	1085	1001	PokeOff1
200:	1087	42	LightOn2
201:	1095	1012	PokeOn2
202:	1095	22	Feed2
203:	1095	32	LightOff2
204:	1096	41	LightOn1
205:	1097	31	LightOff1

206:	1098	42	LightOn2
207:	1104	22	Feed2
208:	1104	32	LightOff2
209:	1106	1002	PokeOff2
210:	1109	1012	PokeOn2
211:	1110	41	LightOn1
212:	1117	1002	PokeOff2
213:	1118	42	LightOn2
214:	1119	1011	PokeOn1
215:	1122	1001	PokeOff1
216:	1123	1012	PokeOn2
217:	1124	31	LightOff1
218:	1124	32	LightOff2
219:	1128	1002	PokeOff2
220:	1129	1012	PokeOn2
221:	1132	41	LightOn1
222:	1135	31	LightOff1
223:	1138	1002	PokeOff2
224:	1139	1011	PokeOn1
225:	1142	1001	PokeOff1
226:	1157	42	LightOn2
227:	1158	32	LightOff2
228:	1159	41	LightOn1
229:	1160	1011	PokeOn1
230:	1161	1001	PokeOff1
231:	1162	1011	PokeOn1
232:	1167	21	Feed1
233:	1167	31	LightOff1
234:	1170	1001	PokeOff1
235:	1172	1012	PokeOn2
236:	1173	42	LightOn2
237:	1176	22	Feed2
238:	1176	32	LightOff2
239:	1176	41	LightOn1
240:	1181	42	LightOn2
241:	1182	1002	PokeOff2
242:	1183	1011	PokeOn1
243:	1183	21	Feed1
244:	1183	31	LightOff1
245:	1185	32	LightOff2
246:	1189	42	LightOn2
247:	1190	1001	PokeOff1
248:	1191	1012	PokeOn2
249:	1191	32	LightOff2
250:	1194	1002	PokeOff2
251:	1195	42	LightOn2
252:	1198	1012	PokeOn2
253:	1198	22	Feed2
254:	1198	32	LightOff2
255:	1202	1002	PokeOff2
256:	1205	1011	PokeOn1
257:	1205	41	LightOn1
258:	1209	21	Feed1
259:	1209	31	LightOff1
260:	1210	41	LightOn1
261:	1211	1001	PokeOff1

262:	1212	1011	PokeOn1
263:	1212	21	Feed1
264:	1212	31	LightOff1
265:	1214	71	WNoiseOff
266:	1214	121	EndTrial
267:	1216	125	EndSession

Session File Formats

If you have data in a custom format not supported by our functions, then you need to write a custom load routine. This is not difficult, but there are certain guarantees your code must make in order for it to be compatible with our code. The TSlibrary functions expect certain information about Sessions, all of which must be passed back by your function. After getting all this information, our session loading routines will place it into the appropriate places in the Experiment structure.

There are a large number of fields in Session designed to accommodate whatever peripheral data may be stored in your Session files. Much of it is not required, and can be set to the empty array [] without any problems.

The job of your function is to take a filename and get as much of this information as is available out of the file.

Your function must have the following syntax. It takes one argument, the filename to be loaded. It returns:

[SUCCESS , ExperimentID, SubjectID, Phase, Box, MATLABStartDate, Duration, TSdata, Notes, Weight, Program, FileReportedUnits]

All fields are required to be returned by your function.

All fields should default to [] if they are not found in the file, or not supported in your custom data format.

Success - This is a flag which should be true if the load was successful and false if it was a failure.

ExperimentID - This is the identification number of the Experiment. This is not a stringent requirement for your function to meet. However, if you do not provide it, you will recieve a warning, and we will assume that it belongs to the current Experiment. If you provide the number but it does not match the current Experiment, you will receive a warning. We generally assume that you wont load the wrong Experiment's data.

SubjectID - This is the identification number of the Subject this data is for. This is a required field, because if we do not know what subject it is we dont know where to put the Session. If it is empty, it is an error. If the number returned is not listed in this Experiment's list of subjects, that is also an error.

Phase - This is the phase number that this Session represents. This is not stringently required, if the empty matrix is returned you will receive a warning and it will be set to 1.

Box - This is the number of the box in which the session was recorded. This is not stringently required, if the empty matrix is returned you will recieve a warning and it will be set to 1.

MATLABStartDate - This is the date and time at which the Session started. This is a stringent requirement, if it is missing you will recieve an error. This should be a matlab date number, e.g., something returned from the matlab function "datenum". Datenum will accept a vector containing [Y, M, D, H, M, S] (year month date hours minutes seconds), or a string representing a date. See help datenum for more information about this.

Duration - This is a totally optional field representing the duration of the trial, also as a matlab date number. If you do not record this in your data file, set it to [], and we will determine it by using the last time stamp in the TSdata and converting that time into seconds.

TSdata - This field should contain the actual raw data of your Experiment, in 2 column time stamped data format. Most of our functions assume this data format, but if you really have a need for it, you could use a different format and most things would continue to work just fine, especially if you still have a double matrix with time-stamps in the left column. In reality, no checks are made on this field, so you can put your data in whatever form is convenient for you. If you do this, be sure to provide a duration, or else there will be an error.

Notes - This totally optional field should hold a character array or cell string containing notes on this session. This is not critical in any way to the functioning of the TS library, and if you do not store it in your data file, you should return an empty character array (''), and it is easy to add your notes to the Sessions using the TSexperimentbrowser.

Weight - This, like notes, is a totally optional field, which should hold a number representing the weight of the subject at the time of the session. It is easy to add it later, return an [] if you do not want to add it.

Program - This is another optional field meant to keep a record of the program used to collect this data. It should hold a string or cell string. Return '' if you do not want to add it.

FileReportedUnits - This field is used if the datafile has a record of what units its timestamps are in. It is analogous to the InputTimeUnit field in the TSsetloadparameters function. If these 2 values do not match, a warning will be given, and the file will override the load parameter setting.

Your function will not need to access the Experiment structure at all, nor will it need to access or apply the loading parameters. You should NOT perform any unit conversions in this function, TSloadsessions will take care of that.

You should throw warnings and errors if the data file is flawed.

In general you can craft this function around the custom format you choose to use. MATLAB has numerous routines for reading different formats, and is capable of reading just about anything. As long as you can get the Subject ID, Date, and TSData at a bare minimum, you can pretty much ignore the other fields and either hardcode them as empty or [1]. In fact, it is possible to encode the Subject ID, Date, and even other fields into the filename, and have the file just contain ascii, or binary, TSdata.

That said, these other fields are here for later convenience, and we encourage you to use a format that has enough information so that years later you will be able to figure out what the file was and what session it represents. If you provide them, then they will be readily accessible from the Experiment Browser window, and your records of the Experiment will be that much more complete.

Standard Format

We have developed a standard session file format which TSlib has built in support for. The assumption is that the datafile forms a 2 column matrix of integer or floating point values.

At the start, there is a header section containing peripheral information in the first column, with identifier flags in the second column attached to each number. Then there is a row of two 0's, which forms a separator between the header and the actual TSdata, which continues until the end of the file.

Example:
<Month>, 1
<Date>, 2

```
<Year>,       3
<Hours>,      4
<Minutes>,    5
<Seconds>,    6
<Experiment>, 7
<Subject>,    8
<Phase>,      9
<Box>,        10
<TimeUnit>,   11
<Weight>,     12
0,            0
... (tsdata)...
...       ...
...       ...
```

In this example, <…> should be replaced by their numeric values.

Note that the rows in the header can be in any order (except for the seperator obviously) because they are identified by the integers 1-12 in the right hand column. They can even be absent, and functions that support this format will set them to [].

TSlib has 3 functions that support this format: TSloadstdtab, TSloadstdxls, and TSloadstdcsv

Standard Tab = Standard format above, using Tab delimiting for columns and end-lines for rows
Standard CSV = Standard format above, with Comma delimiting for columns, end-lines for rows
Standard XLS = Standard format above, in an Excel spreadsheet file.

The functions themselves are quite simple: Here is the source for TSloadstdxls, as an example for the reader.

```
function [ SUCCESS , ExperimentID, SubjectID, Phase, Box, MATLABStartDate, Duration,
TSdata, Notes, Weight, Program, FileReportedUnits ] = TSloadstdxls ( filename )
%
% Assumptions:
%   File is tab delimited
%   Dlm file forms a 2 column matrix
%   First 12 rows are various session fields with indicator numbers on the
%   right column which tell the program which field it is. These do not
%   have to be in any specific order. At the end, a row of 0,0 seperates
%   the tsdata from the codes.
%   Month,     1
%   Date,      2
%   Year,      3
%   Hours,     4
%   Minutes,   5
%   Seconds,   6
```

```
%   Experiment, 7
%   Subject,    8
%   Phase,      9
%   Box,        10
%   TimeUnit,   11
%   Weight,     12
%   0,          0
%   ... (tsdata)...
%   ...         ...
%   ...         ...
%
%   Only Subject and date are really required. Other things are not
%   strictly required but are good to have. TimeUnit is strictly optional, and
%   represents the unit the time stamps are measured in, using
%   seconds e.g. if your unit is 50ths of a seconds, this field should be
%   .02. If this field is not provided the load parameters will be used. If
%   this field does not match the load parameters setting, a warning will
%   be passed and the data file will override the load parameters setting.
%
%   Weight is strictly optional and if your apparatus does not take note of
%   this, you are encouraged to use the TSexperimentbrowser to enter this
%   data at the end of the Experiment.

SUCCESS = 0;

raw = xlsread(filename);

seperator = find(raw(:,2) == 0, 1, 'first');

%Default values for everything
month = [];
date = [];
year = [];
hours = [];
minutes = [];
seconds = [];
ExperimentID = [];
SubjectID = [];
Phase = [];
Box = [];
FileReportedUnit = [];
Notes = '';
Weight = [];
Program = '';
FileReportedUnits = [];
Duration = [];
```

```
TSdata = [];

%Parse the header
x = 1;
while x < seperator
    switch raw(x,2)
        case 1
            month = raw(x,1);
        case 2
            date = raw(x,1);
        case 3
            year = raw(x,1);
        case 4
            hours = raw(x,1);
        case 5
            minutes = raw(x,1);
        case 6
            seconds = raw(x,1);
        case 7
            ExperimentID = raw(x,1);
        case 8
            SubjectID = raw(x,1);
        case 9
            Phase = raw(x,1);
        case 10
            Box = raw(x,1);
        case 11
            FileReportedUnit = raw(x,1);
        case 12
            Weight = raw(x,1);
    end
    x = x + 1;
end

%Throw warnings if date parts are missing
if isempty(month)
    warning('TSload:NoMonth', 'This sesssion data file %s did not provide the month of the
session.', filename);
end

if isempty(date)
    warning('TSload:NoDate', 'This sesssion data file %s did not provide the date of the session.',
filename);
end

if isempty(year)
```

```
   warning('TSload:NoYear', 'This sesssion data file %s did not provide the year of the session.',
filename);
end

if isempty(hours)
   warning('TSload:NoHour', 'This sesssion data file %s did not provide the hour of the session.',
filename);
   hours = 12;
end

if isempty(minutes)
   warning('TSload:NoMinute', 'This sesssion data file %s did not provide the minutes of the
session.', filename);
   minutes = 0;
end

if isempty(seconds)
   seconds = 0;
end

MATLABStartDate = datenum(year, month, date, hours, minutes, seconds); %Get
matlabstartdate from time variables

TSdata = raw(seperator+1:end, :);   %TSdata is everything after the seperator
TSdata = sortrows(TSdata, 1);       %make sure it sorted by timestamp

TSdata(all((TSdata == circshift(TSdata, 1))'),:) = []; %eliminate repeats

SUCCESS = 1;
```

MEDPC format

We have developed a custom MedPC format which is suitable for storing large quantities of time stamped data. MedPC does not support a 2 column output, so we encode the timestamps and event codes into one integer, by multiplying the times by 10^5. Since event codes range from 11-99999, there cannot be a collision here. MedPC has their own header syntax so this function parses theirs.

It is recommended, if you want to use this format, that you use MedPC format 3 “stripped, C array only” variant apprended all, and store the encoded TSdata in the ‘C’ array so that it will be the only thing besides the headers stored to the file.

TSloadMEDPC can then load this session data correctly. It parses the headers assuming that row 1 contains Month, row 2 contains Date, row 3 contains Year, etc. as specified by the MedPC standard.

Then, it removes all lines less than 10^5 and keeps the remainder as the TSdata lines. This will remove the header, which actually is reinserted periodically by MEDPC and needs to be detected and removed unless you take care of it this way. The column vector of encoded information is then expanded into full 2 column TSdata, duplicates codes are checked for and removed, and then this is returned.

You should see the MedPC Users Manual for more information about how to do this, or see (ref).