Developer's Guide

22nd July 2021

Version 5.1.1

by the PsPM team[1]:

Dominik R Bach, Giuseppe Castegnetti, Laure Ciernik, Samuel Gerster, Saurabh Khemka, Christoph Korn, Samuel Maxwell, Tobias Moser, Philipp C Paulus, Ivan Rojkov, Matthias Staib, Eshref Yozdemir, Dadi Zhao and collaborators

[1]If you have comments on or error corrections to this documentation, please send them to the PsPM team or post them on: bachlab.org/pspm

Contents

1 General

1.1 Data files: General structure

In PsPM the data is saved in mat-files. Each file contains two variables:

- `infos` A struct variable with general infos
- `data` A cell array with a cell for each channel.

The cells contain a struct with channel specific infos and data. The structs have the mandatory fields:

- `infos.duration` (in seconds)
- `data{n}.header`
  - `data{n}.header.chantype` (as defined in the settings)
  - `data{n}.header.sr` (sample rate in 1/second, or timestamp units in seconds)
  - `data{n}.header.units` (data units, or 'events')
  - `data{n}.data` (actual data)

Additionally, a typical file contains the optional infos:

- `infos.sourcefile`
- `infos.importfile`
- `infos.importdate`
- `infos.sourcetype`
- `infos.recdate`
- `infos.rectime`

Some data manipulation functions (in particular, pspm_trim) update infos to record some file history.

1.2 How to add a new import data type

1.2.1 Add function

Function name: pspm_get_xxx (where xxx is the data type name).

Format:

```
[sts, import, sourceinfo] = pspm_get_xxx(datafile, import)
```

The function needs to take an import job and add, for each job, fields

- .data - the actual data for this channel (column vector)
- .sr - the sample rate for this channel (only if .autosr enabled in pspm_init)

optional fields

- .marker - for marker channels (timestamps or continuous, see pspm_get_marker)
- .markerinfo – optional, see pspm_get_marker
- .minfreq - minimum frequency for pulse channels
- .units - if data units are defined by the recording software
- sts: -1 if import is unsuccessful

sourceinfo: contains information on the source file, with field

- .chan - a cell of string descriptions of the imported source channels, e. g. names, or numbers any optional fields that will be added to infos.source (e. g. recording date & time, and others)

Notes for multiple blocks: file formats that support multiple block storage within one file can return cell arrays import{1:blkno} and sourceinfo{1:blkno}; PsPM will save individual files for each block, with a filename 'pspm_fn_blkox.mat'.

1.2.2 Add information to settings

The file pspm_init contains a block that defines possible import data types. Add a new field here

```
% Description of data type
% -------------------------------------------
defaults.import.datatypes(1) = ...
struct('short', 'xxx', ... % short name for internal purposes
'long', 'Datatype description', ... % long name for GUI
'ext', '*', ... % data file extension
'funct', @pspm_get_xxx, ... % import function
'chantypes', {{defaults.chantypes.type}}, ... % allowed channel types
'chandescription', 'channel', ... % description of channels for GUI
'multioption', 1, ... % import of multiple channels for GUI
'searchoption', 1, ... % allow channel name search for GUI
'automarker', 0, ... % marker not stored in separate channel
'autosr', 1); % sample rate automatically assigned
```

Good to know:

- the “long” definition is used in the GUI – make sure it’s readable
- if no event channels can be imported, change .chantypes
- if channels have searchable names in the import file, set .searchoption = 1

- if no channel number needs to be assigned for the marker channel, set .automarker = 1
- if sample rate is contained in import file and determined during import, set .autosr = 1
- if you need external functions – put them into a folder in the 'import' subdirectory and add/remove this path within the pspm_get_xxx function

1.3 How to add a new channel type

1.3.1 Add function

Function name: pspm_get_xxx (where xxx is the channel type)

Format:

```
[sts, data] = pspm_get_channeltype(import)
```

data: data cell of structure readable by pspm_load_data

Good to know: for event channels, use the function pspm_get_events to convert various event formats into time stamps (see pspm_get_marker or pspm_get_hb as an example)

1.3.2 Add information to settings

Add information on the new channel type and import function to

```
defaults.chantypes(k).type = 'xxx';% channel type name
defaults.chantypes(k).import = @pspm_get_xxx; % conversion function
defaults.chantypes(k).data = 'xxx'; % wave or events
```

1.4 How to add a new GLM type

1.4.1 Add information to settings (Example SCR)

```
defaults.glm(1) = ...
struct('modality', 'scr', ... % modality name
'cbf', struct('fhandle', @pspm_bf_scrf, 'args', 1), ...
% default basis function/set
'filter', struct('lpfreq', 5, 'lporder', 1, ...
'hpfreq', 0.05, 'hporder', 1, 'down', 10, 'direction', 'uni'));
% default filter settings
```

1.4.2 Add default basis function

Function name: pspm_bf_xxx

Function arguments: vector of arguments, first element is time resolution, further arguments as defined in defaults.glm(n).cbf.args

1.5 Warning IDs in PsPM

1.5.1 General

- invalid_input
- invalid_channeltype
- nonexistent_file
- channel_not_contained_in_file
- obsolete_function
- not_allowed_channeltype
- invalid_data_structure
- no_matching_channels
- unknown_action
- missing_data
- out_of_range

1.5.2 Function specific

pspm_load1

- not_saving_data

pspm_interpolate

- option_disabled

pspm_trim

- marker_out_of_range

pspm_find_channel

- multiple_matching_channels

pspm_find_sounds

- no_marker_chan

- no_sound_chan

pspm_get_scr

- no_conversion_constant

pspm_pp

- invalid_freq

pspm_prepdata

- no_low_pass_filtering
- downsampling_failed
- nonint_sr

pspm_get_timing

- invalid_vector_size
- event_names_dont_match
- no_numeric_vector
- no_integers

pspm_down

- rate_below_minimum

2 List of data formats

2.1 Supported Channel types

Data format	SCR	ECG	Heart Rate	Heart Beat	Heart Period	Respiration	Pupil Size	Marker	Custom	Sound channel	Pulse oxymeter	Gaze x/y, l/r
CED Spike	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Matlab	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Text	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Biopach AcqKnowledge (≤ v3.9.0)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Biopac AcqKnowledge (exported)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Labchart (any Version, Windows only)	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Labchart exported (≤ v7.1)	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Labchart exported (≥ v7.2)	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
VarioPort	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Biograph Infiniti (exported)	✓			✓		✓						
Mindmedia Biotrace (exported)	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Brain Vision	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Windaq (wdq)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
Observer XT compatible	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NeuroScan	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
BioSemi	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Eyelink							✓	✓	✓			✓
European Data Format	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Philips Scanphyslog		✓				✓		✓	✓		✓	
SMI							✓	✓	✓			✓
ViewPoint							✓	✓	✓			✓

2.2 Further settings

Data format	Datatype	File extension	Manufacturer	Import multiple channels	Search channel names	Automarker	Ask for sampling rate
CED Spike	spike	.smr	CED	✓	✓		
Matlab	mat	.mat		✓			✓
Text	txt	.txt		✓	✓		
Biopach AcqKnowledge (≤ v3.9.0)	acq	.acq	Biopac	✓	✓		
Biopac AcqKnowledge (exported)	acqmat	.mat	Biopac	✓	✓		
Labchart (any Version, Windows only)	labchart	.adicht	ADInstruments	✓	✓	✓	
Labchart exported (≤ v7.1)	labchartmat_ext	.mat	ADInstruments	✓	✓	✓	✓
Labchart exported (≥ v7.2)	labchartmat_in	.mat	ADInstruments	✓		✓	
VarioPort	vario	.vpd	Becker MediTec	✓	✓	✓	
Biograph Infiniti (exported)	biograph	.txt	Thought Technology				
Mindmedia Biotrace (exported)	biotrace	.txt	MindMedia			✓	
Brain Vision	brainvision	.eeg	BrainProducts	✓	✓	✓	
Windaq (wdq)	windaq	.wdq	Dataq	✓			
Observer XT compatible	observer	.any	Noldus	✓	✓		
NeuroScan	cnt	.cnt		✓	✓	✓	
BioSemi	biosemi	.bdf		✓	✓	✓	
Eyelink	txt	.asc		✓		✓	
European Data Format	edf	.edf	European Data Format	✓	✓	✓	
Philips Scanphyslog	txt	.log	Philips	✓			
SMI	txt	.txt	SensoMotoric Instruments	✓	✓	✓	
ViewPoint	txt	.txt	Arrington Research	✓	✓	✓	

Note: Automarkers means no channel number has to be specified because markers are always at the same place.

3 GUI

Contributed by Gabriel GrÃ€ni.

3.1 Matlabbatch: Getting started

1. Add the trunk folder to the matlab path
2. Type pspm_init into the command window (after the execution of the command the folders pspm_cfg and matlabbatch should be added to the matlab path)
3. Start matlabbatch by the typing cfg_ui into the command window
4. If the item PsPM exists in the menu bar of matlabbatch you can skip steps 5 to 7 and continue at step 8
5. Select –> File –> Add Application
6. Navigate to the folder pspm_cfg on the left hand side of the window and select the file pspm_cfg.m on the right hand side –> Press the button Done
7. A new item, called PsPM, will appear in the upper menu bar.
8. By selecting PsPM the desired action can be selected (at the moment, there is only Data Preparation –> {Import, Trim} available)

3.1.1 Example Function: Trim

This example demonstrates how matlabbatch can be used to execute a function. For all other functions matlabbatch behaves in the same manner.

- Select a file by pressing the Select Files Button (under Datafile)
- Select Reference and choose an item in the lower part of the window
- Fill in the desired values in the fields which are marked with "<-X"
- After you have chosen a file and filled in all values correctly, you will see a green arrow on the upper left part of the window
- By pressing on the green arrow the selected file will be trimmed according to the filled in values

3.2 Matlabbatch: How to

3.2.1 Preliminaries

- Add folder of matlabbatch to the matlab path
- Add first application and then load the batch in order to execute a function

3.2.2 Some notes for creating a new application

- Leafs (items) are specified first
- Assigning child items to .val or .values fields of their parent items
- Root node of a tree is specified last
- Some examples of items:
  - cfg_item:

```
item1= cfg_item; % Defines generic configuration item
item1.name= 'Def 1'; % The display name
item1.tag = 'def1'; % The name appearing in the harvested job
% structure. This name must be unique
% among all items in the val field of the
% superior node
item1.val = {true}; % Value of item (optional)
item1.help = {'Help...'}; % Help text
```

  - cfg_entry:

```
entry1 = cfg_entry; % Defines entry configuration item
entry1.name = 'Input';
entry1.tag = 'input';
entry1.strtye = 'r'; % Type of values which can be entered
entry1.num = [1 1]; % Expected dimension of the input
entry1.help = {'Help...'};
```

  - cfg_choice:

```
choice = cfg_item; % Defines choice configuration item
choice.name = 'Choice';
choice.tag = 'choice';
choice.values = {item1, entry1}; % Defines which items will be
% selectable in the choice menu.
choice.help = {'Help...'};
```

  - cfg_exbranch:

```
fct = cfg_exbranch; % Defines the branch that has information
% about how to run this module fct.name = 'Trim';
fct.tag = 'trim';
fct.val = {choice}; % The items that belong to this branch.
% All items must be filled before this
% branch can run or produce virtual
% outputs
```

```
fct.prog = @cfg_run_fct; % A function handle that will be called
% with the harvested job to run the
% computation
trim.vout = @cfg_vout_fct; % A function handle that will be
% called with the harvested job to
% determine virtual outputs
trim.help = {Help...'};
```

  - There exists a number of other item classes. Here is a list of the most important classes: cfg_item, cfg_entry, cfg_choice, cfg_menu, cfg_exbranch, cfg_files, cfg_branch, cfg_repeat

    For more information call the help function in matlab (e.g. help help cfg_item)

- Note:
  The inputs to each module have to be described in a tree-like structure.
  During data entry, there is no way to change the tree structure based on input data.
  Add application to the configuration tree by default

3.2.3 Add application to the configuration tree by default

In the following it is shown how an application can be added to the menu bar of matlabbatch by default (without adding it every time matlabbatch is started)

- Start matlabbatch and add the appliaction cfg_confgui in the folder matlabbatch/cfg_confgui
- Put Generate code into the Module list by selecting ConfGUI –> Generate code in the menu bar
- Fill out all the input fields on the right side:
  - Output filename: This file will contain the whole menu structure, validity constraints and links to run time code of the appliaction.
  - Output directory: All files which are created by the ConfGUI will be stored into this directory (chose a directory which is added to the matlab path)
  - Root node of config: Name of the root node of the appliaction's configuration tree
  - Options:
    1. Create Defaults File: Yes
    2. Create mlbatch_appcfg File: Yes
    3. Create Code for addpath(): No
- Finally press the green arrow on the upper left side of the batch editor
- As no error occurred 3 new files ({Output filename}.m, {Output filename}_def.m, cfg_mlbatch_appcfg.m) should be created and added into the folder {Output directory}.
- Each time matlabbatch is started, it will search for any cfg_mlbatch_appcfg.m file (this file contains the names of the configuration files) and will add the corresponding application to the batch editor.

3.2.4 Add modules to module list

Example: Module Import and Trim will be added to the module list

```
arg1 = 'scr.prep.import_data';
arg2 = 'scr.prep.trim';
mod_cfg_id1 = cfg_util('tag2mod_cfg_id',arg1);
mod_cfg_id2 = cfg_util('tag2mod_cfg_id',arg2);
cjob = cfg_util('initjob');
mod_job_id1 = cfg_util('addtojob', cjob, mod_cfg_id1);
mod_job_id2 = cfg_util('addtojob', cjob, mod_cfg_id2);
cfg_util('harvest', cjob, mod_job_id1);
cfg_util('harvest', cjob, mod_job_id2);
cfg_ui('local_showjob', cfg_ui, cjob);
```

3.2.5 Changes

- In the function private/cfg_onscreen at line 36 figure(fg); is commented out in order to prevent the appearance of the GUI for a short time if the function cfg_ui('Visible', 'off') is called.

3.3 Matlabbatch: changing help texts and fieldnames

3.3.1 File structure of matlabbatch GUI

There exist two files per function: 1 configuration file and 1 run file. The configuration file defines the structure of the corresponding function in the matlabbatch GUI whereas the run file firstly gathers all entered values and secondly calls the corresponding SCR function. Both types of files are located in the subfolder pspm_cfg. The name of a configuration or a run file consists of two parts. The prefix of a configuration filename is called pspm_cfg_ whereas the filename of a run file begins with pspm_cfg_run. The second part of the filename is named after the function name (eg. for the function pspm_import.m -> pspm_cfg_import.m, pspm_cfg_run_import.m).

3.3.2 Edit help texts and fieldname

In order to change any help text or fieldname in a matlabbatch GUI function the corresponding configuration file has to be opened. For each item in a matlabbatch GUI function a struct variable which contains several struct fields is defined in the configuration file.

- Help text The field .help defines the help text of the item which can be edited in order to change the help text. As soon as matlabbatch has been closed and opened again, the changes in the help text will be visible in matlabbatch GUI.

- Fieldname The fieldname of an matlabbatch GUI item is defined by the struct field .tag . In case a fieldname of an item should be changed be careful to verify if no other item, which has the same root node, hold the same fieldname. Otherwise matlabbatch will not work properly. After the fieldname of an item has been changed the run file (pspm_cfg_run_functionname.m) of the corresponding function has to be adapted as well in order to ensure that the function call in the run file is done properly.

3.4 Recommendations for GUI development

3.4.1 Mlapp

Mlapp is the new format for developing GUI in Matlab. The old GUIDE based GUI will lose support from Mathworks in a future release. Therefore, it is recommended to move from the old GUIDE to the new Mlapp.

The .mlapp file can be created natively through the new Matlab GUI guide. Alternatively, it can be generated by converting the classic GUIDE based .fig file through the feature *migration*.

3.4.2 Style

The new GUI is currently using the colour #7f2534 for stylishing. The main typeface for UI design is Verdana, which is supported in Windows, macOS and Linux.

4 Test Environment

Contributed by Linus RÃŒttimann & Tobias Moser.

4.1 Unittest: General implementation

In PsPM the Matlab Unit Testing Framework is used for testing of functions. For each tested function there is a Matlab class with the name 'functionname_test', which contains the unittests for that specific function. Additionally there is a documentation page for each of the test classes, where information about the unittests can be found.

To run the unittests of a test class, an object of the class has to be created:

testCase = functionname_test

where 'testCase' is an arbitrary object name and 'funtionname_test' is the name of a test class. Then all the unittest that are contained in the test class can be run with:

testCase.run

A specific unittest can be run with:

testCase.run('unittest_name')

Remember that a new test class object must be generated each time the test class has been changed.

4.2 Parameterized test classes

Parmeterized test classes is a feature provided by the Matlab test case class. A test class is parameterized when it has

- Test parameters defined (within the property section)
- Test methods implementing the defined test parameters

Each function implementing test parameters will be called multiple times with each possible parameter combination (which is determined by Matlab). Thus parameterized classes allow to write single tests for different parameter combinations. If one of the following test cases is a parameterized test class, it will be mentioned accordingly.

4.3 Testcases: pspm_align_channels_test

4.3.1 Information

Testclass: pspm_align_channels_test
Function: [sts, data, duration] = pspm_align_channels(data, induration)

4.3.2 Setup

This test uses data stored in `ImportTestData/ecg2hb/tpspm_s102_s1.mat`

4.3.3 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings given invalid inputs.

Lower optional duration

Function name: lower_optional_duration(this)
Description: Passes an optional duration that is less than the maximum duration of all channels in the input to pspm_align_channels.

Tests:

1. Assert that lower optional duration has no effect on the output.
2. Check if all of the returned channels have the same duration.

Same optional duration

Function name: same_optional_duration(this)
Description: Passes an optional duration that is equal to the maximum duration of all channels in the input to pspm_align_channels and does the exact same checks as in lower duration case.

Higher optional duration

Function name: higher_optional_duration(this)
Description: Passes an optional duration that is higher than the maximum duration of all channels.

Tests:

1. Assert that durations of all returned channels is the same as the passed optional duration.

Max duration is passed in marker channel

Function name: max_duration_is_given_in_events(this)
Description: Passes the maximum duration in marker channel to pspm_align_channels.

Tests:

1. Assert that all returned channels are aligned to the maximum duration passed in marker channel.

Various case checks

Function names:

1. only_one_channel_longer_others_same(this)
2. only_one_channel_shorter_others_same(this)
3. increasing_channel_lengths(this)
4. two_same_others_shorter(this)

Description: In each of these cases check if the returned channels have the same duration that is equal to the maximum duration of all input channels.

4.4 Testcases: pspm_butter

4.4.1 Information

Testclass: pspm_butter_test
Function: [sts, b, a] = pspm_butter(order, freqratio, pass)

4.4.2 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid and if the Signal processing toolbox is installed.

Tests:

Input	Expected warning
pspm_butter() [no input]	ID:invalid_input
pspm_butter(1,1,'abc') [pass not equal to 'high' or 'low']	ID:invalid_input
pspm_butter(2,1) ['Signal processing toolbox is missing' #1]	ID:toolbox_missing
pspm_butter(1,1) ['Signal processing toolbox is missing' #2]	ID:toolbox_missing

4.5 Testcases: pspm_bf_test

4.5.1 Information

Testclass: pspm_bf_test
Function: [bs, x] = pspm_bf_<specific function name>

4.5.2 Setup

This test class is parameterized.

Method setup parameters

These parameters define which function should be tested**.**

Basis function	Specifies the basis functions to test (without the 'pspm_bf_' prefix). The current basis function to test is then called via this.bf();

Test parameters

These are parameters which define what kind of data or option should be passed to each basis function.

Time res log	Specifies for the basic test different time resolutions (argument 'td') which a basis function should be able to handle (as long as td <= duration). The values are logarithmic and have to be translated before passed to the basis function.

4.5.3 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
this.bf() [no parameters]	ID:invalid_input
this.bf(dur+1) [pass 'td' > duration of function]	ID:invalid_input
this.bf(0) [invalid time resolution]	ID:invalid_input

Basic

Function name: test_basic(this, time_res_log)
Description: Test for different requirements to verify whether the current basis function is valid or not.

Tests:

1. Test with td = 0.1, verify that no warning is issued and determine the duration
2. Test with td = 0.01 and check if the new duration is equal to the duration calculated before.
3. Test if function runs through without warning and that the time vector begins at <= 0.
4. Test if the function runs through without warning with td = 10^time_res_log (as long as td < duration)

4.6 Testcases: pspm_convert_unit

4.6.1 Information

Testclass: pspm_convert_unit_test
Function: [sts, converted] = pspm_convert_unit(data, from, to)

4.6.2 Setup

Constants

- inch_to_cm = 2.54

4.6.3 Testcases

Invalid input

Function name: invalid_input(testCase)
Description: Pass invalid from or to metrics and check if warnings are issued.

Valid input

Function name: valid_input(this)
Description: Pass various valid inputs and compare results to manually calculated ones.

Tests:

1. If empty input data is passed, result is also empty.
2. Convert single cm value to m.
3. Various unit conversion checks:

   (a) Conversion between same units (cm to cm)
   (b) mm to km and km to mm conversions
   (c) inch to cm conversions

4. Negative value conversions

5. Convert single dimensional array with multiple elements.

6. Convert each element in 3D array.

4.7 Testcases: pspm_ecg2hb

4.7.1 Information

Testclass: pspm_ecg2hb_test

Function: [sts,pt_debug] = pspm_ecg2hb(fn, chan, options)

4.7.2 Setup

Constants

- testdata{0}.chan_struct = struct('nr', 1, 'name', 'ecg');
- testdata{0}.filename = 'ImportTestData\ecg2hb\test_ecg77.mat';
- testdata{0}.num_channels = 1
- testdata{1}.chan_struct = struct('nr', 3, 'name', 'ecg');
- testdata{1}.filename = 'ImportTestData\ecg2hb\tpspm_s102_s1.mat';
- testdata{1}.num_channels = 5
- backup_suffix = '_backup';
- options = struct('semi', 0);

4.7.3 Testcases

Invalid input arguments

Function name: invalid_input(this)

Description: Pass invalid input arguments and check if the warnings are as expected.

Tests:

Input	Expected warning
pspm_ecg2hb() [no arguments]	ID:invalid_input
pspm_ecg2hb(1) [invalid file name]	ID:invalid_input
pspm_ecg2hb(this.fn, 'bla') [invalid channel (text)]	ID:invalid_input
pspm_ecg2hb(this.fn, 1) [invalid channel type]	ID:not_allowed_channeltype
o.twthresh = 'bla'; pspm_ecg2hb(this.fn, this.chan.nr, o) [invalid twthresh (text)]	ID:invalid_input
o.minHR = 202; pspm_ecg2hb(this.fn, this.chan.nr, o) [invalid minHR (> default_maxHR)]	ID:invalid_input
o.minHR = 202; o.maxHR = 19; pspm_ecg2hb(this.fn, this.chan.nr, o) [invalid minHR > maxHR]	ID:invalid_input
o.maxHR = 19; pspm_ecg2hb(this.fn, this.chan.nr, o) [invalid maxHR (< default_minHR)]	ID:invalid_input
o.debugmode = 5; pspm_ecg2hb(this.fn, this.chan.nr, o) [invalid debugmode (not in [0,1])]	ID:invalid_input
o.semi = 5; pspm_ecg2hb(this.fn, this.chan.nr, o) [invalid semi (not in [0,1])]	ID:invalid_input

Valid input arguments

Function name: valid_input(this)

Description: Pass valid input arguments and check if there are no warnings.

Tests:

Input	Expected warning
pspm_ecg2hb(this.fn, this.chan.nr, this.options)	-
pspm_ecg2hb(this.fn, this.chan.name, this.options)	-
this.test_added_data()	-

4.7.4 Other Methods

Test for added data

Function name: test_added_data()

Description: Check if added hb channels show an expected behaviour.

Tests (for each channel):

Tested Value	Expected Value
Sampling rate	1
Unit	'events'
Channel type	'hb'
Amount of data points in data	> 1
Heartbeat indices are monotonically increasing	True
Maximum number of heartbeats per second	< 5
Data is distributed equally (standard deviation)	< 2s
Time between end of recording and last data point	< 60s

4.8 Testcases: pspm_filtfilt

4.8.1 Information

Testclass: pspm_filtfilt_test
Function: y = pspm_filtfilt(b,a,x)

4.8.2 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_filtfilt() [no input]	ID:invalid_input
pspm_filtfilt([1:10],[1:20],[1:10]) [data length less than 3 times filter order]	ID:invalid_input

4.9 Testcases: pspm_find_channel

4.9.1 Information

Testclass: pspm_find_channel_test

Function: chan = pspm_find_channel(headercell, chantype)

4.9.2 Testcases

Invalid input arguments

Function name: invalid_inputargs(this)

Description: Checks for warnings, if the input arguments are invalid.

Setup:

```
headercell = {'heart', 'scr', 'pupil'};
```

Tests:

Input	Expected warning
pspm_find_channel('str','scr') [no headercell]	ID:invalid_input
pspm_find_channel(headercell, 'str')	ID:not_allowed_channeltype
pspm_find_channel(headercell, 4) [no string chantype]	ID:invalid_input

Valid Input Arguments

Function name: valid_inputargs(this)

Description: Checks for correct return value if the input arguments are valid

Setup:

```
headercell = {'heart', 'scr', 'pupil', 'mark', 'gsr', 'eda'};
```

Tests:

Input	Exp. Output	Expected warning
pspm_find_channel(headercell, 'pupil')	3	
pspm_find_channel(headercell, 'resp')	0	ID:no_matching_channels
pspm_find_channel(headercell, 'scr')	-1	ID:multiple_matching_channels
pspm_find_channel(headercell, {'mark', 'str', 'bla'})	4	
pspm_find_channel(headercell, {'call', 'str', 'me'})	0	no matching channel, but no warning
pspm_find_channel(headercell, {'scr', 'gsr', 'eda'})	-1	multiple matching channels, but no warning

4.10 Testcases: pspm_extract_segments

4.10.1 Information

Testclass: pspm_extract_segments_test
Function: [sts, out] = pspm_extract_segments(varargin)

4.10.2 Setup

This test class is parameterized. For manual mode tests, the test data is generated by the function itself and when needed, files will be written to testdatafile<variable_nr>.mat. For auto mode tests, the test data must be in `ImportTestData/fitted_models` folder with names as specified in the tests.

Test parameters

These are parameters which define what kind of data should be passed to `pspm_extract_segments` in auto mode tests and which options should be set.

- nan_output: This option defines whether the user wants to output the NaN ratios of the trials for each condition. If so, we values can be printed on the screen (on MATLAB command window) or written to a created file.
- nan_ratio: Defines ratio of NaN values in the generated test data
- nr_trail: Number of trails in the generated test data

4.10.3 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.
Tests:

Input	Expected warning
pspm_extract_segments()	ID:invalid_input
pspm_extract_segments('a','b')	ID:invalid_input
pspm_extract_segments('manual',fn,0)	ID:invalid_input
pspm_extract_segments('manual',struct('a',10),0,timing)	ID:invalid_input
pspm_extract_segments('manual',[1,3],logical(32),timing)	ID:invalid_input
pspm_extract_segments('manual',fn,'a',timing)	ID:invalid_input
pspm_extract_segments('manual',fn,{'a'},timing)	ID:invalid_input
pspm_extract_segments('auto',{1})	ID:invalid_input
pspm_extract_segments('auto','some')	ID:invalid_input

Test manual mode with indicated length

Function name: test_manual_length(this,nr_trial,nan_ratio)
Desctiption: Checks for equality of produced segments by pspm_extract_segments with manually computed segments

Tests:

1. Generate segments form test data.
2. Test if function call wirked WarningFree
3. Test if variable 'segments' existis in output
4. Test if correct number of segments were produced
5. Test each segment holds correct data

Test manual mode with durations

Function name: test_manual_duration(this,nr_trial,nan_ratio)
Desctiption: Checks for equality of produced segments by pspm_extract_segments with manually computed segments

Tests:

1. Generate segments form test data.
2. Test if function call wirked WarningFree
3. Test if variable 'segments' existis in output
4. Test if correct number of segments were produced
5. Test each segment holds correct data

Test auto mode with GLM using marker onsets

Function name: `test_auto_mode_glm_with_markers(this)`
Description: Runs `pspm_extract_segments` with a particular GLM model stored in `ImportTestData/fitted_models` and compares the results to manually calculated results.

Tests:

1. Test if length of the returned cell array (from now on called `segments`) is the same as the number of conditions
2. Test if shape of data arrays in each element of `segments` agrees with the passed durations and number of onsets.
3. Test if `segments.trial_idx` agrees with input data.
4. Test if statistics calculated manually from `segments.data` is the same as `segments.mean` and `segments.std`.
5. Compute each statistic field in each element of `segments` manually using the input data and compare the results to `segments`.

Test auto mode with GLM using second onsets

Function name: `test_auto_mode_glm_with_seconds(this)`
Description: Do the exact same tests as in `test_auto_mode_glm_with_markers` but this time using seconds to specify onsets.

Test auto mode with DCM

Note: Since in DCM case onsets are calculated using trial start and end seconds of DCM trials, there is no second/marker distinction in DCM test.
Function name: `test_auto_mode_dcm(this)`
Description: Runs `pspm_extract_segments` with a particular DCM model stored in `ImportTestData/fitted_models` and compares the results to manually calculated results. In order to get meaningful condition statistic information this test function assigns the same trial name to certain groups of trials.

Tests: Do the exact same tests as in `test_auto_mode_glm_with_markers` by adapting the computation steps to DCM case.

4.11 Testcases: pspm_find_sounds

4.11.1 Information

Testclass: pspm_find_sounds_test
Function: [sts, infos] = pspm_find_sounds(file, options)

4.11.2 Setup

This test class is parameterized. The test data is generated by the function itself and when needed, files will be written to testdatafile<variable_nr>.mat.

Test parameters

These are parameters which define what kind of data should be passed to pspm_find_sounds and which options should be set.

Channel output	Specifies whether 'all' found markers or only 'corrected' markers should be returned.
Max delay	Varies the max delay option and defines how far away a marker at most can be.
Min delay	Varies the min delay option and defines how far away a marker at least should be.
Threshold	Defines the minimum size of a marker to be recognized as a marker event. Passed in percent of the maximum amplitude of the recorded data.
Resample	Defines whether the function should resample (and interpolate) the data to a higher sample rate in order to get more exact marker findings.
Channel action	Defines whether a newly created marker channel should replace the existing marker channel or should be added as a new marker channel.

4.11.3 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_find_sounds(”)	ID:file_not_found
pspm_find_sounds(fn) [invalid pspm file]	ID:invalid_input
pspm_find_sounds(fn) [pspm file without a ’snd’ channel]	ID:no_sound_chan
pspm_find_sounds(fn, o) [invalid values for positive integer fields]	ID:invalid_input
pspm_find_sounds(fn, o) [invalid values for positive numeric fields]	ID:invalid_input
pspm_find_sounds(fn, o) [invalid values for logic fields]	ID:invalid_input
pspm_find_sounds(fn, o) [invalid channel ids for channel fields]	ID:out_of_range
pspm_find_sounds(fn, o) [enabled diagnostics without a marker channel]	ID:no_marker_chan
pspm_find_sounds(fn, o) [invalid values for channelaction]	ID:invalid_input
pspm_find_sounds(fn, o) [invalid values for roi]	ID:invalid_input
pspm_find_sounds(fn, o) [maxdelay < mindelay]	ID:invalid_input

Test add channel

Function name: test_add_channel(this, channeloutput, max_delay, resample, channelaction)
Description: Test add channel with different options. Diagnostics is always enabled, Channel output, Max delay, Resample and Channel action are varied. Once pspm_find_sounds is complete, the function tests if the returned data has the expected format.

Tests:

1. Generate data with channel ’snd’ and ’marker’; and count amount of reference markers
2. Set
   (a) options according to test parameters
   (b) diagnostics to 1
3. Test if function runs through without warning
4. Test if returned data has the correct format
5. Test if channels has been added or replaced
6. Test if added channel has correct amount of data

Test region count

Function name: test_region_count(this)
Description: Test region of interest in combination with expected sound count.

Tests:

1. Generate data with channel 'snd' and 'marker'
2. Test if function finds the function finds all markers in the whole file
3. Test if function finds all the markers in the whole file with initial threshold 1
4. Test if function finds half of the markers in half of the file

Test threshold

Function name: test_threshold(this, threshold)
Description: Vary the threshold option and test whether the functions returnes the expected data.

Tests:

1. Generate data with channel 'snd' and 'marker'
2. Set
   (a) threshold according to test parameter
   (b) diagnostics to 1
3. Test if function runs through without warning
4. Test if returned data has the correct format

Test plot

Function name: test_plot(this, threshold)
Description: Test if the plot functions returne the expected data and runs through without warning.

Tests:

1. Generate data with channel 'snd' and 'marker'
2. Set
   (a) plot to 1
   (b) diagnostics to 1
3. Test if function runs through without warning
4. Test if returned data has the correct format

4.12 Testcases: pspm_find_valid_fixations

4.12.1 Information

Testclass: pspm_find_valid_fixations_test
Function: [sts, out_file] = pspm_find_valid_fixations(fn, options)

4.12.2 Setup

This test class is parameterized. The test data is generated by the function itself and when needed, files will be written to testdatafile<variable_nr>.mat.

Test parameters These are parameters which define what kind of data should be passed to pspm_find_valid_fixations and which options should be set.

Distance	Used for gaze validation; defines the distance between eyes and screen.
Aspect used	Used for gaze validation; defines the aspect ratio set in the software.
Aspect actual	Used for gaze validation; defines the aspect ratio of the hardware.
Screen size	Used for gaze validation; defines the size of the screen in inches.
Eyes	Is used for data generation and tells the function for which eyes data should be generated.
Channel action	Defines whether to 'add' or 'replace' existing channels.
Newfile	Defines whether to create a new file or extend the existing file.
Overwrite	Defines whether to overwrite the existing file or not.
Interpolate	Defines whether to interpolate NaN values in validated channels or not.
Missing	Defines whether to create a channel which holds information about which positions have been set to NaN (and may have been interpolated afterwards).
Work eye	Defines which eyes should be used for fixation validation.
Work chans	Defines which channels should be set to NaN during invalid fixations.

4.12.3 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_find_valid_fixations()	ID:invalid_input
pspm_find_valid_fixations('a')	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.validate_fixations]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.box_degree]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.screen_settings]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [missing fields for options.screen_settings]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.aspect_actual]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.aspect_used]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.bitmap]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.display_size]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.display_size]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.fixation_point]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.channel_action]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.newfile]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.overwrite]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.interpolate]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.missing]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid eyes]	ID:invalid_input
pspm_find_valid_fixations(fn, options) [invalid options.channels]	ID:invalid_input

Test work chans

Function name: test_work_chans(this, work_chans)
Description: Tests whether the option 'channels' actually works on the specified channels or not.

Tests:

1. Generate data with distance 500, aspect_used 16:9, aspect_actual 4:3, screen_size 20 and eyes ’lr’
2. Set options with
   (a) overwrite = 1
   (b) channels = work_chans
   (c) channel_action = ’add’
3. Test if function runs through without warning
4. Test if sts equals 1
5. Test if specified work_chans are added as new processed channels

Test work eye

Function name: test_work_eye(this, work_eye)
Description: Test whether the option ’eyes’ actually works on the specified eyes or not.

Tests:

1. Generate data with distance 500, aspect_used 16:9, aspect_actual 4:3, screen_size 20 and eyes ’lr’
2. Set options with
   (a) overwrite = 1
   (b) eyes = work_eye
   (c) channel_action = ’add’
3. Test if function runs through without warning
4. Test if sts equals 1
5. Test if specified eyes have been processed accordingly and test if not specified eyes have ignored.

Test missing

Function name: test_missing(this, missing)
Description: Test whether for each a a new missing channel is created if missing is specified as true.

Tests:

1. Generate data with distance 500, aspect_used 16:9, aspect_actual 4:3, screen_size 20 and eyes ’lr’

2. Set options with
    (a) overwrite = 1
    (b) missing = missing
    (c) channel_action = 'add'
3. Test if function runs through without warning
4. Test if sts equals 1
5. Depending on the status of 'missing' test if there are any missing channels or if there is no missing channel

Test interpolate

Function name: test_interpolate(this, interpolate)
Description: Test whether data is interpolated during periods which are set to NaN by the function.

Tests:

1. Generate data with distance 500, aspect_used 16:9, aspect_actual 4:3, screen_size 20 and eyes 'lr'
2. Set options with
    (a) overwrite = 1
    (b) interpolate = interpolate
    (c) channel_action = 'add'
3. Test if function runs through without warning
4. Test if sts equals 1
5. Depending on the status of 'interpolate' test whether there are some NaN values or if NaN periods have been interpolated accordingly.

Test overwrite

Function name: test_overwrite(this, overwrite)
Description: Test if files are overwritten, if specified with 'overwrite' option.

Tests:

1. Generate data with distance 500, aspect_used 16:9, aspect_actual 4:3, screen_size 20 and eyes 'lr'
2. Set options with

    (a) overwrite = 1
    (b) interpolate = interpolate
    (c) channel_action = 'add'

3. Test if function runs through without warning
4. Test if sts equals 1
5. Test if file has been overwritten or not (tests, if there are any new channels).

Test channel action

Function name: test_channel_action(this, channel_action)
Description: Test if channels are added or replaced (according to channel_action).

Tests:

1. Generate data with distance 500, aspect_used 16:9, aspect_actual 4:3, screen_size 20 and eyes 'lr'
2. Set options with
    (a) overwrite = 1
    (b) channel_action = channel_action
3. Test if function runs through without warning
4. Test if sts equals 1
5. Test if channels have been added or replaced (tests, if there are any new channels).

Test newfile

Function name: test_newfile(this, newfile)
Description: Test whether the output is written to a newfile or to the input file.

Tests:

1. Generate data with distance 500, aspect_used 16:9, aspect_actual 4:3, screen_size 20 and eyes 'lr'
2. Set options with
    (a) overwrite = 1
    (b) if newfile enabled
        i. search for new file name
        ii. set options.newfile to new file name
    (c) if newfile is disabled, set options.newfile to ''

3. Test if function runs through without warning
4. Test if sts equals 1
5. Test if returned outputfile equals the specified newfile or not (depending on the value of 'newfile')

Test gaze validation

Function name: test_gaze_validation(this, distance, screen_size, aspect_actual, aspect_used, eyes)
Description: Test whether gaze validation is done correctly.

Tests:

1. Generate data with the according function parameters
2. Iterate to returned degree values generated by the generation function
   (a) set function options
      i. overewrite = 1
      ii. validate_fixation =1
      iii. screen_settings and distance to function call settings
      iv. missing = 1
   (b) depending on the specified degree, test whether function runs through without warnings or not
   (c) load outputfile and test if (according to degree expectation) gaze validation has been done or not

Test bitmap validtion

Function name: test_bitmap_validation(this, distance, resolution, eyes)
Description: Test whether bitmap validation is done correctly.

Tests:

1. Generate data with the according function parameters
2. Iterate to returned bitmaps generated by the generation function
   (a) set function options
      i. missing = 1
   (b) ddepending on the specified number of valid fixations in the bitmap, test whether function runs through without warnings or not
   (c) load outputfile and test if (according to bitmap expectation) bitmap validation has been done or not

4.13 Testcases: pspm_get_ecg

4.13.1 Information

Testclass: pspm_get_ecg_test
Function: [sts, data] = pspm_get_ecg(import)

4.13.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'ecg'
4. Test if data.header.units is equal import.units
5. Test if data.header.sr is equal import.sr

4.14 Testcases: pspm_get_events

4.14.1 Information

Testclass: pspm_get_events_test
Function: [sts, import] = pspm_get_events(import)

4.14.2 Testcases

Check warnings

Function name: check_warnings(this)
Description: Checks for warnings, if the field '.markers' is missing or contains invalid content.

Tests:

Input	Expected warning
Missing marker field	ID:nonexistent_field
import.marker = 'foo'	ID:invalid_field_content

Timestamps

Function name: timestamps(this)
Description: Checks for correct output if the input is timestamp data

Tests:

1. Test if 'sts' is equal 1.
2. Test if the length of the output data is equal to the length of the input data

Continuous

Function name: continuous(this)
Description: Checks for correct output if the input is continuous data

Tests:

1. Perform three tests with different settings

   Tests:

   (a) Test if 'sts' is equal 1.
   (b) Test if the length of the field 'markerinfo' is equal to the length of the output data.
   (c) Test if the length of the output data is equal to the expected number of pulses in the input data.

   Settings:

   (a) flank = 'both' (default)
   (b) flank = 'both' & data offset 50
   (c) flank = 'ascending'
   (d) flank = 'descending'
   (e) inverted input signal
   (f) signal with angular flanks
   (g) check with

2. Additional test for setting (b): Test if data offset has been removed in the output data.
3. Additional test for setting (c) and (d): Test if positions returned by output data correspond to flank changes in the input data.
4. Test if markerinfo is not set if it has been set before.

4.15 Testcases: pspm_get_eyelink

4.15.1 Information

Testclass: pspm_get_eyelink_test
Function: [sts, data] = pspm_get_eyelink(import)

4.15.2 Methods

set_import_values

Function : [import_struct, channel typles] = set_import_values(this)
Description: Helperfunction, which creates an import data set and the expected channel data set

verify_basic_data_structure

Function name: verify_basic_data_structure(this, data, sourceinfo, channel_types)
Description: Tests if the returned data structure is valid and match a given expected pattern.

Tests:

1. Test if all channels are numeric
2. Test if recorded time and date have a valid format
3. Test if blink channels have correct unit
4. Test if pupil channels have either 'diameter' or 'area' as unit
5. Test if channels labeled with 'position' have unit 'pixel'
6. Test if channels labeled with 'blink' have unit 'blink'

4.15.3 Testcases

test_multi_session

Function name: test_multi_session(this)
Description: Test if the returned data structure fits into the pattern of a multi session data set.

Tests:

1. Calls 'set_import_values(this)' to get import data set and expected channel data set
2. passses returned sets to 'verify_basic_data_structure()'

test_two_eyes

Function name: test_two_eyes(this)
Description: Test if the returned data structure fits into the pattern of a two eyes data set.

Tests:

1. Calls 'set_import_values(this)' to get import data set and expected channel data set
2. passes returned sets to 'verify_basic_data_structure()'

test_one_eye

Function name: test_one_eye(this)
Description: Test if the returned data structure fits into the pattern of a one eye data set.

Tests:

1. Creates an import data set and the expected channel data set an pass it to 'verify_basic_data_structure()'

test_track_dist

Function name: test_track_dist(this)
Description: Test if the returned data structure fits into the pattern of a two eyes data with eyelink_trackdist set.

Tests:

1. Calls 'set_import_values(this)' to get import data set and expected channel data set
2. owerwrites some import data and channel data
3. pass returned sets to 'verify_basic_data_structure()'

4.16 Testcases: pspm_get_hb

4.16.1 Information

Testclass: pspm_get_hb_test
Function: [sts, data] = pspm_get_hb(import)

4.16.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'hb'
4. Test if data.header.units is 'events'
5. Test if data.header.sr is 1

4.17 Testcases: pspm_get_hr

4.17.1 Information

Testclass: pspm_get_hr_test
Function: [sts, data] = pspm_get_hr(import)

4.17.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'hr'
4. Test if data.header.units is equal import.units
5. Test if data.header.sr is equal import.sr

4.18 Testcases: pspm_get_marker

4.18.1 Information

Testclass: pspm_get_marker_test
Function: [sts, data] = pspm_get_marker(import)

4.18.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'marker'
4. Test if data.header.units is 'events'
5. Test if data.header.sr is 1

4.19 Testcases: pspm_get_pupil

4.19.1 Information

Testclass: pspm_get_pupil_test
Function: [sts, data] = pspm_get_pupil(import)

4.19.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'pupil'
4. Test if data.header.units is equal import.units
5. Test if data.header.sr is equal import.sr

4.20 Testcases: pspm_get_resp

4.20.1 Information

Testclass: pspm_get_resp_test
Function: [sts, data] = pspm_get_resp(import)

4.20.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'resp'
4. Test if data.header.units is equal import.units
5. Test if data.header.sr is equal import.sr

4.21 Testcases: pspm_get_scr

4.21.1 Information

Testclass: pspm_get_scr_test
Function: [sts, data] = pspm_get_scr(import)

4.21.2 Testcases

There are three test functions. One for the case that no transfer parameters are defined, one for the case that the transfer parameters are defined in a struct and one for the case that they are defined in a .mat file. They are all performing the following tests, plus eventually some individual tests

Tests:

1. Test if 'sts' is equal 1.
2. Test if the field data.data exists
3. Test if the field data.data is not empty
4. Test if the field data.header.units exists
5. Test if the field data.header.sr exists
6. Test if the field data.header.chantype exists
7. Test if data.header.sr is equal import.sr
8. Test if data.header.chantype is 'scr'

No transfer parameters

Function name: no_transferparams(testCase)
Description: Test if all fields are returned correctly, if no transfer parameters are defined.

Additional Tests:

No additional tests

Struct transfer parameters

Function name: stuct_transferparams(testCase)
Description: Test if all fields are returned correctly, if the transfer parameters are defined in a struct.

Additional Tests:

1. Check for warning if the conversion constant (import.transfer.c) is not defined
2. Checks that there are no warnings if import.transfer.Rs or import.transfer.offset is not defined.

File transfer parameters

Function name: file_transferparams(testCase)
Description: Test if all fields are returned correctly, if the transfer parameters are defined in a .mat file.

Additional Tests:

1. Check for warning if the transfer parameter file doesn't exist.

4.22 Testcases: pspm_get_timing

4.22.1 Information

Testclass: pspm_get_timing_test
Function:
[sts, multi] = pspm_get_timing('onsets', intiming, timeunits)
[sts, events] = pspm_get_timing('markervalues', markerinfo, names)
[sts, epochs] = pspm_get_timing('epochs', epochs)
[sts, events] = pspm_get_timing('events', events)

4.22.2 Testcases

Invalid input arguments

Function name: invalid_inputargs(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_get_timing('epochs') [missing input var]	ID:invalid_input
pspm_get_timing('onsets', 'str') [no timeunits var]	ID:invalid_input
pspm_get_timing('foo') [unknown format]	ID:invalid_input
pspm_get_timing('onsets', intiming, 'samples') [two sessions with nonmatching number of conditions]	ID:number_of_elements_dont_match
pspm_get_timing('onsets', intiming, 'samples') [two sessions with nonmatching condition names]	ID:event_names_dont_match
pspm_get_timing('onsets', intiming, 'samples') [intiming.onsets{1} is no numeric vector]	ID:no_numeric_vector
pspm_get_timing('epochs', fn_mat, 'samples') [epochs is not an integer array]	ID:no_integers
pspm_get_timing('markervalues', markerinfo) [no markervalue and no name]	ID:invalid_input
pspm_get_timing('markervalues', markerinfo, markervalue, names) [markervalue is not of numeric type nor a cell array]	ID:invalid_input
pspm_get_timing('markervalues', markerinfo, markervalue, names) [markervalue and names are not of the same length]	ID:invalid_input

Case Epochs

Function name: case_epochs(this)
Description: Checks the function in 'epochs' mode.
Function: [sts, epochs] = pspm_get_timing('epochs', epochs)

Test 1 (matfile input)

Input: mat file with variable: epochs = [1 4; 2 5; 3 6]

Check if sts==1 and if the return value is equal the input array.

Test 2 (spm input)

Input: mat file with variable: onsets{1} = [1 2 3]';onsets{2} = [4 5 6]';

Check if sts==1 and if the return value is equal [onsets{1}, onsets{2}].

Test 3 (textfile input)

Input: textfile with variable: epochs = [1 4; 2 5; 3 6]

Check if sts==1 and if the return value is equal the input array.

Test 4 (matrix input)

Input: matrix: epochs = [1 4; 2 5; 3 6]

Check if sts==1 and if the return value is equal the input array.

Case onsets

Function name: case_onsets(this)
Description: Checks the function in 'onsets' mode.
Function: [sts, multi] = pspm_get_timing('onsets', intiming, timeunits)

Test 1

Input: mat file with the variables:
names = {'name1', 'name2'};
onsets = {[1 2], [3 4]};
pmod.name = {'name3', 'name4'};
pmod.param = {[2 3], [4 5]};
pmod.poly = {2, 2};
save(fn_mat, 'names', 'onsets', 'pmod');

Function call:
[sts, outtiming] = pspm_get_timing('onsets', fn_mat, 'samples');

Tests:
Check if sts==1, if onsets and names are unchanged and if
outtiming.pmod.param == {[2 3], [4 9], [4 5], [16 25]}

Test 2

Input:
mat file with the variables: names = {'name1', 'name2'};
onsets = {[1 2 3], [3 4 5]}; durations = {[3 4 5]', [5 6 7]'};
pmod.name = {'name3', 'name4'};
pmod.param = {[2 3 4], [4 5 6]};
pmod.poly = {2, 1};

Function call:
[sts, outtiming] = pspm_get_timing('onsets', fn_mat, 'samples');

Tests:

Check if sts==1, if onsets,names and durations are unchanged and if
outtiming.pmod.param == {[2 3 4], [4 9 16], [4 5 6]}

Case events

Function name: case_events(this)
Description: Checks the function in 'events' mode.
Function: [sts, epochs] = pspm_get_timing('events', events)

Check the function if input is a one element cell array and a multiple element cell array. Check for warnings (ID:invalid_vector_size) if elements have more than two columns and if not all elements have the same number of rows.

4.23 Testcases: pspm_get_<datatype>

4.23.1 Information

The datatype import functions are all tested in a similar way. The individual testclasses must inherit the class 'pspm_get_superclass', from which they inherit the main test function 'valid_datafile'. They also have to implement the property 'fhandle', which is a function handle to the specific import function.

The tests are performed with the sampledata files that are listed in the SampleDataMasterList.docx file (as at 18.11.2013).

Superclass: pspm_get_superclass

Testclasses: pspm_get_acq_test
pspm_get_acqmat_test
pspm_get_biograph_test
pspm_get_biosemi_test
pspm_get_biotrace_test
pspm_get_brainvis_test
pspm_get_edf
pspm_get_labchartmat_ext_test
pspm_get_labchartmat_in_test
pspm_get_mat_test
pspm_get_obs_test
pspm_get_spike_test
pspm_get_superclass
pspm_get_txt_test
pspm_get_vario_test
pspm_get_eyelink_test

Function: [sts, import, sourceinfo] = pspm_get_<datatype>(datafile, import)

4.23.2 Notes

4.23.3 Setup

define testcases

In this method the testcases are defined and the testdata is generated (if needed). Each testcase is a cell in the cellarray 'testcases'. Each testcase has the following fields:

- .pth: the path to the samplefile
- .import: the input variable

For datatypes which support blocks there has to be an additional field:

- .numofblocks

4.23.4 Testcases

Valid datafile

Function name: valid_datafile(this)

Description: The main test function, for tests with valid inputdata. It tests all testcases equally.

Tests:

1. Test if 'sts' is equal 1.
2. If the datatype supports blocks, test if the number of blocks is correct.
3. Test if number of elements of the returned 'import' variable is correct.
4. Test if each importjob has a field 'data', that is a numeric vector.
5. Test if each importjob has a field 'sr', that is a number.
6. Test if each importjob has a field 'type'.
7. Test if all event importjobs have a field 'marker'.
8. Test if all importjobs have duration below 1h.
9. Test if all importjobs have a samplerate between 1 and 10000 for continuous channels or between 10^-6 and 1 for timestamp channels.

invalid datafile

Function name: invalid_datafile(this)
Description: The main test function, for tests with invalid inputdata.

Tests:

If the datatype supports multiple channels: Check for warning when trying to import a channel, that is not contained in the file ('ID:channel_not_contained_in_file').

4.24 Testcases: pspm_get_acq

In this section we describe the testcases specific to pspm_get_acq apart from generic pspm_get tests.

4.24.1 Information

Testclass: pspm_get_acq_test
Function: [tss, import, sourceinfo] = pspm_get_acq(datafile, import)

4.24.2 Testcases

get_acq should return the exact same data as Acqknowledge exported mat file

Function name: get_acq_returns_same_data_as_acqknowledge_exported_mat(this)
Description: The data obtained by using pspm_get_acq should be identical with the data obtained by using export .mat file functionality in Acqknowledge software.

Tests:

1. Load data stored in `ImportTestData/acq/impedance_acq.acq` and `ImportTestData/acq/impedance_mat.m` separately. Then compare the first channel of `impedance_mat` with the first channel of data obtained by calling `pspm_get_acq` on `impedance_acq`.

4.25 Testcases: pspm_glm

4.25.1 Information

Testclass: pspm_glm_test
Function: glm = pspm_glm(model, options)

There are seven testcase functions. One invalid input arguments test and test 1 to 6. Tests 1 to 5 are of the same kind. There are one or multiple testcases per test function, have a look at the testcase description for more information. In these tests only Kronecker delta functions are used as basis functions, furthermore all conditions, pmods and nuisance regressors are pairwise orthogonal. The data is also not down sampled and not filtered. With these limitations it's easy to calculate the data vectors and the expected stats. For each testcase it is then tested:

- If numel(glm.names) has the expected value.
- If numel(glm.stats) has the expected value.
- If glm.stats has the expected value (with a tolerance of 1%).

In test 6 the default basis functions are used, and not all conditions and pmods are orthogonal. The data is down sampled and low and high pass filtered. In exchange the stats are not tested for correct values, just for the correct number of elements. The properties 'shiftbf' and 'norm' are TestParameters, which means that this testclass is parameterized. All functions implmementing these parameters (Test 1 to Test 5) are called several times with all the different values and combinations of the mentioned parameters.

4.25.2 Testcases

Invalid input arguments

Function name: invalid_input (this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_glm(model) [no timeunits field]	ID:invalid_input
pspm_glm(model) [no timeunits var]	ID:invalid_input
pspm_glm(model) with model.timeunits = 'foo' [no valid timeunits field]	ID:invalid_input
pspm_glm(model) with model.timing = zeros(10,2) [no valid timing field]	ID:invalid_input
pspm_glm(model) with model.modality = 'foo' [no valid modality field]	ID:invalid_input
pspm_glm(model) with model.channel = 'scr' [no valid channel field]	ID:invalid_input
pspm_glm(model) with model.norm = 'no' [no valid norm field]	ID:invalid_input
pspm_glm(model) with model.filt.down = 'none' [filt.down is not numeric]	ID:invalid_input
pspm_glm(model) with model.bf.fhandle = 'foohandle' [non existing bf]	ID:invalid_fhandle
pspm_glm(model) with numel(model.datafile) != numel(model.timing)	ID:number_of_elements_dont_match
pspm_glm(model) with model.missing is struct [non valid missing field]	ID:invalid_input
pspm_glm(model) with numel(model.datafile) != numel(model.missing)	ID:number_of_elements_dont_match
pspm_glm(model) with model.nuisance is struct [non valid nuisance field]	ID:invalid_input
pspm_glm(model) with numel(model.datafile) != numel(model.nuisance)	ID:number_of_elements_dont_match
pspm_glm(model) with no R variable in the nuisance file	ID:invalid_input
pspm_glm(model) with R variable in the nuisance file that has not the same length as the datafile	ID:number_of_elements_dont_match

Test 1

Function name: test1(this, shiftbf)
Description: Basic test with one basis function, one session, no nuisance regressors, no missings and one condition. Timeunits are seconds.

Testcases:

1. no pmods
2. one pmod
3. two pmods

Test 2

Function name: test2(this, shiftbf)
Description: Test with one basis function, one session, no nuisance regressors, no missings and two conditions. Timeunits are seconds.

Testcases:

1. no pmods
2. first condition: no pmods; second condition: one pmod
3. first condition: one pmod; second condition: two pmods

Test 3

Function name: test3(this, shiftbf)
Description: Test with one basis function, one session, two nuisance regressors (1Hz cosinus, 1Hz sinus), no missings, one condition and no pmods. Timeunits are seconds.

Testcases:

Only one testcase.

Test 4

Function name: test4(this, shiftbf)
Description: Test with one basis function, two sessions, no nuisance regressors, no missings and one condition.

Testcases:

1. timeunits are seconds
2. timeunits are samples
3. timeunits are markers

Test 5

Function name: test5(this, shiftbf)
Description: Test with two basis functions, one session, no nuisance regressors and one condition. Timeunits are seconds.

Testcases:

1. no missings
2. with missings

Test 6

Function name: test6(this)
Description: Test with default basis function and non-orthogonal conditions and pmods

Testcase:

Default basis functions, no nuisance regressors, no missings, two sessions and two conditions. Timeunits are seconds.

- first condition: two pmods (with pmod(1).poly{1} = 2 and pmod(1).poly{2} = 3)
- second condition: no pmods

Test7:

Function name: test_extract_missing(this, cutoff, nan_percent)
Description: Test with one basis function, one session, no nuisance regressors, no missings and three conditions. Timeunits are seconds.

Testcases: for all combinations of the test parameters cutoff and nan_percent

- glm vector stats_missing has the appropriate length according to the number of conditions
- percentages in glm vector stats_missing contains the expected value
- glm vector stats_exclude has the appropriate length according to the number of conditions
- glm vector stats_exclude contains the expected condistion which should be excluded

4.26 Testcases: pspm_hb2hp

4.26.1 Information

Testclass: pspm_hb2hp_test
Function: [sts, infos] = pspm_hb2hp(fn, sr, chan, options)

4.26.2 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_hb2hp() [no input]	ID:invalid_input
pspm_hb2hp(2) [not a string filename]	ID:invalid_input
pspm_hb2hp('abc') [no sample rate]	ID:invalid_input
pspm_hb2hp('abc','abc') [not a string sample rate]	ID:invalid_input
pspm_hb2hp('abc',2,'abc') [not a numeric chanel]	ID:invalid_input
pspm_hb2hp(files{1},100) [call of pspm_load_data fails]	ID:invalid_input
pspm_hb2hp(files{2}, 100) [not enough points for interp1]	ID:too_strict_limits
pspm_hb2hp(files{3},100,[],options) [pspm_write_channel fails]	ID:invalid_input

4.27 Testcases: pspm_import

4.27.1 Information

Testclass: pspm_import_test
Function: outfile = pspm_import(datafile, datatype, import, options)

4.27.2 Testcases

Invalid input arguments

Function name: invalid_inputargs(ths)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Test No.	Input	Expected warning
1	pspm_import(datafile, datatype) [no import variable]	ID:invalid_input
2	pspm_import(datafile, datatype, 'foo') [no cell/struct import var.]	ID:invalid_input
3	pspm_import(datafile, 'foo', import) [invalid channeltype]	ID:invalid_channeltype
4	pspm_import(5, datatype, import) [no char filename]	ID:invalid_input

Invalid import variable structure

Function name: invalid_import_struct(this)
Description: Checks for warnings, if the structure of the import variable is invalid.

Tests:

Test No.	Input	Expected warning
1	Multiple channel, though not supported	ID:invalid_import_struct
2	Not allowed channeltype	ID:invalid_import_struct
3	No sr given, though autosr is not supported	ID:invalid_import_struct
4	Nonexistent file	ID:nonexistent_file

One datafile

Function name: one_datafile(this)
Description: Checks the function, if datafile is a string (import of one datafile) and all inputs are correct. The outfile is checked with the pspm_load_data function. The tests are performed with a spike samplefile and a labchartmat_in samplefile (to check the handling of blocks).

Multiple datafiles

Function name: multiple_datafiles(this)
Description: Checks the function, if datafile is a cell array of strings (import of multiple datafiles) and all inputs are correct. The outfiles are tested with the pspm_load_data function.

4.28 Testcases: pspm_interpolate

4.28.1 Information

Testclass: pspm_interpolate_test
Function: [sts, outdata] = pspm_interpolate(indata, options)

4.28.2 Setup

This test class is parameterized. The test data is generated by the function itself and when needed, files will be written to datafile<variable_nr>.mat.

Test parameters

These are parameters which define what kind of data should be passed to pspm_interpolate and which options should be set.

Amount	Specifies how many elements indata (for pspm_interpolate) should have.
Datatype	Specifies what type of data should be generated. • struct - a valid data struct will be generated • inline - a numeric vector will be generated • file - a valid scr file will be generated • all - all types will sequentially be generated until amount is reached
Chans	If datatype is not inline this specifies how many and which type of data channels the generated data should have. In a second field it also defines which of these channels should be interpolated (this will be passed later in options.channels).
Nan method	Specifies how NaN values will be put into the data. • start - range is 1+offset:<random number before the center> • center - range is <radnom number before the center>:<random number after the center> • end - range is <random number after the center>:end-offset The offset is 1 if 'extrap' is not defined. This is needed because if there is no data at the end or beginning of the data, the function is unable to interpolate (unless extrapolation is activated).
Extrap	Is either true or false and activates or deactivates the extrapolation.
Interp method	Specifies the interpolation method.
Newfile	True or false and tells the function to either create a file or add the data as new channel.
Overwrite	True or false and tells the function to either overwrite an existing file or not.
Replace channel	True or false and tells the function to either replace the given channels with the interpolated data or to add the interpolated data as new channel.

4.28.3 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Test No.	Input	Expected warning
1	pspm_interpolate() [no arguments]	ID:missing_data
2	pspm_interpolate({{}}) [data is not char, struct, numeric]	ID:invalid_input
3	pspm_interpolate({}) [data empty]	ID:missing_data
4	pspm_interpolate(struct()) [invalid struct]	ID:invalid_data_structure
5	pspm_interpolate(invalid_data) [file which does not exist]	ID:nonexistent_file
6	pspm_interpolate(valid_data, options) [options.channels is larger than valid_data]	ID:invalid_input
7	pspm_interpolate(valid_data, options) [options.channels is not numeric]	ID:invalid_input
8	pspm_interpolate(valid_data, options) [options.method is invalid]	ID:invalid_input
9	pspm_interpolate(valid_data, options) [options.newfile is invalid]	ID:invalid_input
10	pspm_interpolate(valid_data, options) [options.extrapolate is invalid]	ID:invalid_input
11	pspm_interpolate(valid_data, options) [options.overwrite is invalid]	ID:invalid_input
12	pspm_interpolate(valid_data, options) [options.dont_ask_overwrite is invalid]	ID:invalid_input
13	pspm_interpolate(valid_data, options) [options.replace_channels is invalid]	ID:invalid_input
14	pspm_interpolate(invalid_data, options) [try to interpolate an events channel]	ID:invalid_channeltype
15	pspm_interpolate(invalid_data) [try to interpolate with nan from beginning and without extrapolation]	ID:option_disabled
16	pspm_interpolate(invalid_data, options) [try to interpolate with nan from beginning and with extrapolation]	ID:out_of_range
17	pspm_interpolate(invalid_data) [try to interpolate with nan from end and without extrapolation]	ID:option_disabled
18	pspm_interpolate(invalid_data, options) [try to interpolate with nan from end and with extrapolation]	ID:out_of_range

Test datatypes

Function name: test_datatypes(this, datatype, amount, chans)
Description: Tries to interpolate with different datatypes, amount of data, channels.

Tests:

1. Generate data with datatype, amount, 'center', chans, false
2. Test if function issues no warnings
3. Test if sts is 1
4. Test if size of outdata equals the size of the data
5. Test if channels to be interpolated have no more NaNs
6. Test if channels not to be interpolated still contain NaNs

Test interpolation variations

Function name: test_interpolation_variations(this, interp_method, extrap, nan_method)
Description: Tries to interpolate with different interpolation methods while varying options.extrapolate and the nan_method.

Tests:

1. Generate data with 'inline', 1, nan_method, {{'scr'}, []}, extrap
2. Test if function issues no warnings
3. Test if sts is 1
4. Test if size of outdata equals the size of the data
5. Test if data has no more NaNs

Special case: When extrapolation is on and nan_method is 'start' and interp_method is 'previous' or nan_method is 'end' and interp_method is 'next'. This should issue a warning because this is not possible (e.g. interpolate with previous value when first NaN value is at the beginning of the data set).

1. Generate data as above
2. Test if function issues a warning.

Test no nan

Function name: test_no_nan(this)
Description: Test whether function works even if there is nothing to interpolate.

Tests:

1. Generate data struct() with pspm_test_data_gen()
2. Test if function issues no warnings
3. Test if sts is 1
4. Test if size of outdata equals the size of data
5. Test if outdata equals data
6. Test if data has no NaNs

Test write

Function name: test_write(this, newfile)
Description: Vary the option newfile and test whether new file is created correctly or data is correctly added to a new channel.

Tests:

1. Generate data with 'file', 2, 'center', {{'scr', 'scr', 'scr'}, [1,3]} , false
2. Test if function issues no warnings
3. Test if sts is 1
4. Test if size of outdata equals the size of data
5. Test if outdata does not equal data

New files only:

1. Test if new file exists
2. Load old and new file and test if size of data is equal
3. Verify that interpolated channels in the new file are NaN free

Added to existing file only:

1. Test if all returned values are numeric (new channel ids)
2. Verify that the added channels are NaN free
3. Test if added channels match the size of the original data channels

Test overwrite

Function name: test_overwrite(this, overwrite)
Description: Vary overwrite and test whether files are overwritten or not.

Tests:

1. Generate data with 'file', 2, 'center', {{'scr', 'scr', 'scr'}, [1,2,3]}, false
2. Create files with expected filenames
3. Test if function issues no warning
4. Test if sts is 1
5. According to overwrite test if file has been overwritten or not

Test replace channel

Function name: test_replace_channel(this, replace_channels)
Description: Vary replace_channel and test wether channels are overwritten or not.

Tests:

1. Generate data with 'file', 2, 'center', {{'scr', 'scr', 'scr'}, [1,2,3]} , false
2. Test if function issues no warnings
3. Test if sts is 1
4. Test if size of outdata equals the size of data
5. Test if outdata does not equal data
6. According to replace_channel test whether returned channel ids correspond to replaced channels or correspond to added channels.

4.28.4 Other methods

Generate data

Has all of the Test parameters as parameter implemented and accordingly generates the data. It calls put nan to insert NaN values into the data. The generated data is returned as data to the calling function. Also all return values are stored in the property testdata (for cleanup data).

Cleanup data

Sits in MethodTeardown and is called once the test class has finished all tests. It then removes all the datafiles which can be found in the property 'testdata'.

Verify NaN free

Helper function to verify whether the data is NaN free or not. It copes with two states. Either a channel should have been interpolated, then it shouldn't contain any NaN values or a channel should not have been interpolated, then the channel should still contain NaN values.

4.29 Testcases: pspm_load1

4.29.1 Information

Testclass: pspm_load1_test
Function: [sts, data, mdltype] = pspm_load1(fn, action, savedata, options)

4.29.2 Setup

The datafile fn is referring to a datafile which was generated with pspm_load1_test.generate_testdata(this).
The function is part of the test object and generates models for all of the available model types
(defined in settings.first). The models are created with data generated with pspm_testdata_gen.
Two files belong to each model: model_<modeltype><variable nr.>.mat (fn) and dummy_<modeltype><variable_nr>.mat
(dfn). The model file on the one hand is the actual model file while on the other hand, the
dummy file is a copy of the model file, used by the test to manipulate the test data.

Generated aquisition data (pspm_testdata_gen)

```
data{1}.chantype = 'scr';
data{2}.chantype = 'hb';
```

The duration of the channels is 100s.

Generated GLM model

```
model.timing{1}.names = {'a';'b';'c'};
model.timing{1}.onsets = {[10, 20, 30], [15, 25, 35], [18, 28, 38]};
```

Generated DCM & SF model

```
model.timing{1} = [10,20; 23,38; 40,70;];
model.condition{1}.name = {'a';'b'};
model.condition{1}.index = [1;2];
```

4.29.3 Testcases

Invalid model structure (general)

Function: invalid_model_tructure_general(this)
Description: Tries to pass invalid data structures, and tests for certain warnings. Applys to all available modeltypes.

Tests:

Input	Expected warning
empty model file	ID:invalid_data_structure
missing field 'modelfile'	ID:invalid_data_structure
missing field 'modeltype'	ID:invalid_data_structure
missing field 'modality'	ID:invalid_data_structure
missing field 'stats'	ID:invalid_data_structure
missing field 'names'	ID:invalid_data_structure

Invalid model structure (specific)

Function: invalid_model_structure_general(this)
Description: Tries to pass invalid data structures, and tests for certain warnings. Model specific.

Tests for GLM:

Input	Expected warning
field 'stats' is not an n x 1 vector	ID:invalid_data_structure
unequal amount of numbers and parameters in field 'stats'	ID:invalid_data_structure
options.zscored = 1 & action = 'cond'	ID:invalid_input

Tests for DCM & SF:

Input	Expected warning
unequal size for fields in 'trlnames' and rows in 'stats'	ID:invalid_data_structure
missing field 'trlnames'	ID:invalid_data_structure
unequal size for fields in 'names' and columns in 'stats'	ID:invalid_data_structure
action = 'recon'	ID:invalid_input

Tests for DCM:

Input	Expected warning
options.zscored = 1 & pspm_load1(dfn, 'none', {}, options)	ID:invalid_input
options.zscored = 1 & pspm_load1(dfn, 'cond', {}, options)	-
options.zscored = 1 & pspm_load1(dfn, 'stats', {}, options)	-

Tests for GLM & SF:

Input	Expected warning
options.zscored = 1 & pspm_load1(dfn, 'cond', {}, options)	ID:invalid_input

Action 'none'

Function: test_action_none(this)
Description: Test for all modeltypes if action 'none' matches the expected behaviour.

Tests:

1. Basic function test
2. Test if returned data is empty.

Action 'stats'

Function: test_action_stats(this)
Description: Test for all modeltypes if action 'stats' matches the expected behaviour.

Tests for all:

1. Basic function test
2. Returned data contains field 'stats'
3. Returned data contains field 'names'

Tests for DCM & SF:

1. Returned data contains field 'trlnames'
2. Returned data contains field 'condnames'

Action 'cond'

Function: test_action_cond(this)
Description: Test for all modeltypes if action 'cond' matches the expected behaviour.

Tests for all:

1. Basic function test
2. Returned data contains field 'stats'
3. Returned data contains field 'names'

Tests for DCM & SF:

1. Returned data contains field 'trlnames'
2. Returned data contains field 'condnames'

Action 'recon'

Function: test_action_recon(this)
Description: Test for all modeltypes if action 'recon' matches the expected behaviour.

Tests for GLM:

1. Basic function test
2. Returned data contains field 'stats'
3. Returned data contains field 'names'

Tests for DCM & SF already done in specific structure test.

Action ’savecon’

Function: test_action_savecon(this)
Description: Test for all modeltypes if action ’savecon’ matches the expected behaviour. Generates a number, passes it within the ’savecon’ struct and tests if the number is returned correctly.

Tests:

1. Basic function test
2. Returned data contains field ’con’
3. Field ’con’ contains field ’test’
4. Field ’con.test’ is equal to the randomly generated number

Action ’con’

Function: test_action_con(this)
Description: Test for all modeltypes if action ’con’ matches the expected behaviour. Tests if the in ’savecon’ generated field test is still returned.

Tests:

1. Basic function test
2. Returned data contains field ’con’
3. Field ’con’ contains field ’test’.

Action ’all’

Function: test_action_all(this)
Description: Test for all modeltypes if action ’all’ matches the expected behaviour.

Tests:

1. Basic function test
2. Returned data is not empty.

Action ’save’

Function: test_action_save(this)
Description: Test for all modeltypes if action ’save’ matches the expected behaviour. Test with options.overwrite = 1. Generates random number and writes it into field ’test’ in model structure.

Tests:

1. Basic function test
2. Model structure contains field 'test'
3. Field 'test' in model structure equals to the randomly generated number.

Options

Function: test_options(this)
Description: Test for all modeltypes if options passed with options structure cause the expected behaviour. Does also work with a randomly generated number in <model struct>.test to test whether the data is written or not.

Tests for all:

1. dont_ask_overwrite = 1 & overwrite = 0 returns warning ID:not_saving_data and field 'test' in model struct does not match generated number
2. dont_ask_overwrite = 1 & overwrite = 1 field 'test' in returned model struct does match generated number

Tests for DCM (with dont_ask_overwrite = 1 & overwrite = 1):

1. zscored = 0 & action = 'stats'
   (a) Basic function test
   (b) Returned data.stats is not zscored
2. zscored = 1 & action = 'stats'
   (a) Basic function test
   (b) Returned data.stats is zscored
3. zscored = 0 & action = 'cond'
   (a) Basic function test
   (b) Returned data is different when callng with zscroed = 1 & action = 'cond' (should not zscore, when not specified)

4.29.4 Other methods

Remove testdata

Removes all the test data generated by the test class. It is called once the class is finished with testing.

Basic function test

Is called in each test after the tested function has been called. It does two checks:

- Returned modeltype matches the modeltype stored in the returned model structure
- Returned status (sts) equals 1

4.30 Testcases: pspm_load_data

4.30.1 Information

Testclass: pspm_load_data_test
Function: [sts, infos, data, filestruct] = pspm_load_data(fn, chan)

4.30.2 Setup

If not otherwise declared, the input variable fn is referring to a datafile which was generated with pspm_testdata_gen and consists out of the following channels:

```
data{1}.chantype = 'scr';
data{2}.chantype = 'marker';
data{3}.chantype = 'hr';
data{4}.chantype = 'hb';
data{5}.chantype = 'marker';
data{6}.chantype = 'resp';
data{7}.chantype = 'scr';
```

The duration of the channels is 10s.

4.30.3 Testcases

Invalid input arguments

Function name: invalid_inputargs(testCase)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_load_data [no filename]	ID:invalid_input
pspm_load_data(1) [no char filename]	ID:invalid_input
pspm_load_data(fn, -1) [neg. channel no]	ID:invalid_input
pspm_load_data(fn, 'foobar') [no allowed ch type]	ID:invalid_input
pspm_load_data(fn, foo) [missing field in foo struct]	ID:invalid_input
pspm_load_data(fn, {1}) [invalid channel option]	ID:invalid_input
pspm_load_data(struct) [struct has no infos field]	ID:invalid_input
pspm_load_data(fn, 250) [nonexisting channel]	ID:invalid_input

Invalid datafile

Function name: invalid_datafile(testCase)
Description: Checks for warnings, if the datafile is invalid.

Tests:

Test No.	Input	Expected warning
1	non-existent datafile	ID:nonexistent_file
2	missing 'infos' variable	ID:invalid_data_structure
3	missing 'data' variable	ID:invalid_data_structure
4	missing 'data' field in 'data{2}'	ID:invalid_data_structure
5	missing 'header' field 'data{3}'	ID:invalid_data_structure
6	missing 'sr' field in 'data{7}.header'	ID:invalid_data_structure
7	data{4} is a nx2 vector (instead of a nx1 vector)	ID:invalid_data_structure
8	the length of data{1}.data is incompatible with the duration	ID:invalid_data_structure
9	An entry of data{2}.data is larger than 'duration'	ID:invalid_data_structure
10	data{5} has an non-existent chantype ('scanner')	ID:invalid_data_structure
11	duplicates (9) with struct chan input	ID:invalid_data_structure

Return all channels

Function name: valid_datafile_0(testCase)
Description: Checks the function, if all channels shall be returned (chan = 0).

Return all channels (struct input)

Function name: valid_datafile_1(testCase)
Description: Checks the function, if all channels shall be returned (chan = 0) and the input is a struct.

Return one channel

Function name: valid_datafile_2(testCase)
Description: Checks the function, if only one channel shall be returned (chan = 2).

Return one channel

Function name: valid_datafile_3(testCase)
Description: Checks the function, if multiple channels shall be returned (chan = [3 5]).

Return scr channels

Function name: valid_datafile_4(testCase)
Description: Checks the function, if only the scr channels shall be returned.

Return event channels

Function name: valid_datafile_5(testCase)
Description: Checks the function, if only the event channels shall be returned.

Save data

Function name: valid_datafile_6(testCase)
Description: Checks the function, if data is to be saved (chan struct).

4.31 Testcases: pspm_pp

4.31.1 Information

Testclass: pspm_pp_test
Function: newfile = pspm_pp('median', datafile, n, channelnumber) or newfile = pspm_pp('butter', datafile, freq, channelnumber)

4.31.2 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_pp('butter', 'file') [no freq]	ID:invalid_input
pspm_pp('foo', 'file', 100) [no valid first argument]	ID:invalid_input
pspm_pp('butter', 'file', 19) [freq below 20]	ID:invalid_input

Median test

Function name: median_test(this)
Description: Checks medianfilter functionality

Setup:

Testfile with 3 channels (scr, hb, scr).

Tests:

1. Filter one channel [Input: newfile = pspm_pp('median', testfile, 50, 3)]
   i. Check if sts == 1, when data is loaded with pspm_load_data.
   ii. Check if newfile has the same number of channels as testfile
2. Filter multiple channel [Input: newfile = pspm_pp('median', testfile, 50)]
   i. Check if sts == 1, when data is loaded with pspm_load_data.
   ii. Check if newfile has the same number of channels as testfile

Butterworth filter test

Function name: butter_test(this)
Description: Checks Butterworth filter functionality

Setup:

Testfile with 3 channels (scr, hb, scr).

Tests:

1. Filter one channel [Input: newfile = pspm_pp('butter', testfile, 40, 3)]
   i. Check if sts == 1, when data is loaded with pspm_load_data.
   ii. Check if newfile has the same number of channels as testfile
2. Filter multiple channel [Input: newfile = pspm_pp('butter', testfile, 40)]
   i. Check if sts == 1, when data is loaded with pspm_load_data.
   ii. Check if newfile has the same number of channels as testfile

4.32 Testcases: pspm_prepdata

4.32.1 Information

Testclass: pspm_prepdata_test
Function: [sts, outdata, newsr] = pspm_prepdata(data, filt)

4.32.2 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_prepdata([1 NaN 3], filt) [NaN values in data]	ID:invalid_input
pspm_prepdata([1 2 3]) [no filt variable]	ID:invalid_input
pspm_prepdata(data, filt) [filt has no hporder field]	ID:invalid_input
pspm_prepdata('foo', filt) [no numeric data]	ID:invalid_input
pspm_prepdata(data, filt) [with lpfreq = 'foo' (not valid)]	ID:invalid_input

Hipassfilter test

Function name: hipassfilter_test(this)
Description: Checks hipassfilter functionality (without downsampling)

Setup:

data = rand(1000,1);

filt.sr = 100;
filt.lpfreq = 'none';
filt.lporder = 1;
filt.hpfreq = 20;
filt.hporder = 1;
filt.down = 'none';

Tests:

1. Unidirectional tests [filt.direction = 'uni']
   i. Check if sts == 1
   ii. Check if newsr == filt.sr
   iii. Check if outdata is empty
   iv. Check if length(outdata) == length(data)
2. Unidirectional tests [filt.direction = 'bi']
   i. Check if sts == 1
   ii. Check if newsr == filt.sr
   iii. Check if outdata is empty
   iv. Check if length(outdata) == length(data)

Lowpassfilter test

Function name: lowpassfilter_test(this)
Description: Checks hipassfilter functionality (without downsampling)

Setup:
data = rand(1000,1);
filt.sr = 100;
filt.lpfreq = 40;
filt.lporder = 1;
filt.hpfreq = 'none';
filt.hporder = 1;
filt.down = 'none';

Tests:

Same tests as in hipassfilter_test. Additionally there is a check for a warning if filt.lpfreq is higher (or equal) than the nyquist frequency:

Input	Expected warning
pspm_prepdata(data, filt) [filt.sr = 100; filt.lpfreq = 60]	ID:no_low_pass_filtering

Bandpassfilter test

Function name: bandpassfilter_test(this)
Description: Checks bandpassfilter functionality (without downsampling)

Setup:
data = rand(1000,1);
filt.sr = 200;
filt.lpfreq = 99;
filt.lporder = 1;
filt.hpfreq = 20;
filt.hporder = 1;
filt.down = 'none';

Tests: Same tests as in hipassfilter_test.

Integer samplerate ratio downsampling test

Function name: int_sr_ratio_downsample_test(this)
Description: Checks downsampling functionality, if the ratio between filt.sr and filt.down is an integer.

Setup:

ratio = 2; %ratio between filt.sr and filt.down

filt.down = 100;
filt.sr = ratio
filt.down; filt.lpfreq = 40;
filt.lporder = 1;
filt.hpfreq = 'none';
filt.hporder = 1;
filt.direction = 'uni';

data = rand(filt.sr * 10,1);

Tests:

1. Check if sts == 1
2. Check if newsr == filt.down
3. Check if outdata is empty
4. Check if ratio*length(outdata) == length(data)

4.33 Testcases: pspm_process_illuminance

4.33.1 Information

Testclass: pspm_process_illuminance_test
Function: [sts, out] = pspm_process_illuminance(ldata, sr, options)

4.33.2 Setup

This test class is parameterized. The test data is generated by the function itself and when needed, files will be written to datafile<variable_nr>.mat.

Test parameters

These are parameters which define what kind of data should be passed to pspm_process_illuminance and which options should be set.

bf_dur	Defines the duration of the basis function.
bf_offset	Defines the offset of the basis function.
dur	Defines the duration of the generated dataset.
sr	Defines the samplerate of the generated dataset.
n_times	Defines how many datasets should be generated.
mode	Defines the whether the dataset should be written to a file, kept as inline variable or should be a mix of both. Can be either 'file', 'inline' or 'mixed'
overwrite	Defines whether existing files should be overwritten or not.

4.33.3 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Test No.	Input	Expected warning
1	pspm_process_illuminance() [no arguments]	ID:invalid_input
2	pspm_process_illuminance([]) [empty data]	ID:missing_data
3	pspm_process_illuminance(1:10) [missing samplerate]	ID:invalid_input
4	pspm_process_illuminance(1:10, 'a') [invalid ssamplerate]	ID:invalid_input
5	pspm_process_illuminance({1:10}, 1) [cell, no cell]	ID:invalid_input
6	pspm_process_illuminance(1:10, {1}) [no cell, cell]	ID:invalid_input
7	pspm_process_illuminance({1:10, 10:10}, {1}) [different sized cells]	ID:invalid_input
8	pspm_process_illuminance({1:10, 'a'},{1,2}) [invalid file]	ID:non_existent_file
9	pspm_process_illuminance({1:10, 1:10}, {1, 'a'}) [invalid samplerate]	ID:invalid_input
10	pspm_process_illuminance({1:10}, {1}, 'o') [wrong options]	ID:invalid_input
11	pspm_process_illuminance({1:10}, {1}, opt)[wrong transfer settings]	ID:invalid_input
12	pspm_process_illuminance({1:10}, {1}, opt)[wrong duration]	ID:invalid_input
13	pspm_process_illuminance({1:10}, {1}, opt)[wrong offset]	ID:invalid_input
14	pspm_process_illuminance({1:10}, {1}, opt)[wrong outputfile]	ID:invalid_input
15	pspm_process_illuminance({1:10}, {1}, opt)[format of ldata and opt.fn differs]	ID:invalid_input
16	pspm_process_illuminance({1:10}, {1}, opt)[opt.overwrite is not boolean]	ID:invalid_input

Test options

Function name: test_options(this, sr, dur, bf_dur, bf_offset)
Description: Tries out different combination options to process the generated illuminance data.

Tests:

1. Generate data with sr and dur
2. Set options according to bf_dur and bf_offset
3. Set expected warning according to sr*dur and sr*bf_dur

   (a) expect empty data if sr*dur < 1
   (b) expect invalid_input if sr*bf_dur < 1
   (c) otherwise expect no warning

4. Test if issued warning equals expected warning
5. Test if sts equals expected value
6. Test if amount of data elements of input and output data is equal

Test multi

Function name: test_multi(this, n_times, mode)
Description: Generates n sets of illuminance data and passes it to pspm_process_illuminance.

Tests:

1. Generate data with 10 (sr), 100 (dur), n_times (amount), mode
2. Test if pspm_process_illuminance issues no warning
3. Test if sts is 1
4. For n_times == 1, test if out has 10*100 data points
5. for n_times ~= 1, test if output has same size as input

Test overwrite

Function name: test_overwrite(this, overwrite)
Description: Generate illuminance file and test overwrite behaviour.

Tests:

1. Generate data with 10 (sr), 100 (dur), 1 (amount), 'file'
2. Test if pspm_process_illuminance issues no warning
3. Test if sts equals 1
4. Test if existing file was overwriten or not

4.33.4 Other methods

Generate lx

Has some of the Test parameters as parameter implemented and accordingly generates the lx data. According to the calling arguments the output is a cell of files and data vectors. All generated files will be stored in the property 'datafiles'. They will be removed once all tests have finished.

Cleanup

Located in MethodTeardown and is called once the test class has finished all tests. It then removes all the datafiles which can be found in the property 'datafiles'.

4.34 Testcases: pspm_pulse_convert

4.34.1 Information

Testclass: pspm_pulse_convert_test
Function: wavedata = pspm_pulse_convert(pulsedata, resamplingrate, samplingrate)

4.34.2 Testcases

Invalid input

Function name: invalid_input(testCase)
Description: Pass invalid input arguments and test if the error message is correct.

Tests:

Input	Expected warning
pspm_pulse_convert()	ID:invalid_input
pspm_pulse_convert(10^-3 * (1:10000)')	ID:invalid_input
pspm_pulse_convert(10^-3 * (1:10000)', 10000)	ID:invalid_input

Valid input

Function name: valid_input(testCase)
Description: Pass generated, valid data and test if function issues no warning.

Tests:

1. Test function without downsampling the data
2. Test function with downsampling the data

4.35 Testcases: pspm_ren

4.35.1 Information

Testclass: pspm_ren_test
Function: out_newfilename = pspm_ren(filename, newfilename)

4.35.2 Testcases

Invalid input

Function name: invalid_input (this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_ren('fn') [no newfilename]	ID:invalid_input
pspm_ren({'fn1', 'fn2'}, {'rfn1', 'rfn2', 'rfn3'}) [non same size cell arrays]	ID:invalid_input

Char Valid Input

Function name: char_valid_input (this)
Description: Checks the function if the input variables are of type char. It uses pspm_load_data to check the files.

Tests:

1. Check if out_newefilename = newfilename
2. Check if sts==1 (of pspm_load_data output)
3. Check if the field 'infos.rendata' exists
4. Check if the field 'infos.newname' exists
5. Check if the original file has been deleted

Cell Valid Input

Function name: cell_valid_input (this)
Description: Checks the function if the input variables are of type cell. It uses pspm_load_data to check the files.

Tests:

The inputs are two-element cell arrays. For both elements the same tests as in the char_valid_input function are performed individually.

4.36 Testcases: pspm_resp_pp

4.36.1 Information

Testclass: pspm_resp_pp_test
Function: sts = pspm_resp_pp(fn, sr, chan, options)

4.36.2 Testcases

Regression Test against Revision r660

Function name: compare_results_to_results_obtained_from_r660_version(this)
Description: In r660, there was a bug found in pspm_resp_pp that caused it to crash with index out of bounds error on inputs containing some edgecase. This test specifically checks whether the fixed version returns the same results as the version before the bugfix on data that didn't cause a crash.

Tests:

1. Check if the returned channel types have the same name and ordering
2. Check if the returned data is the same

4.37 Testcases: pspm_split_sessions

4.37.1 Information

Testclass: pspm_split_sessions_test
Properties: expected_number_of_files = 3;
Function: newdatafile = pspm_split_sessions(datafile, markerchannel, options)

4.37.2 Setup

For the tests a testdatafile with three channels is used (duration is 100s). The markerchannel data is:

data = [1 4 9 12 30 31 34 41 43 59 65 72 74 80 89 96]'

Hence if MAXSN=10 & BRK2NORM=3 (default values) the datafiles should be split into 3 files. If different values are being used, update the property 'expected_number_of_files' of the testclass object accordingly.

4.37.3 Testcases

Invalid input

Function name: invalid_input (this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_split_sessions() [no filename]	ID:invalid_input
pspm_split_sessions (2) [no string filename]	ID:invalid_input
pspm_split_sessions ('fn', 'foo') [no numeric marker channel no.]	ID:invalid_input

One datafile

Function name: one_datafile(this)
Description: Checks the function if the variable 'datafile' is of type char (one datafile). The markerchannel number is not assigned explicitly.

Tests:

1. Check if the file has been split into 'expected_number_of_files' files For each output file the following tests are performed:

2. Check if sts == 1, when data is loaded with pspm_load_data.
3. Check if number of channels is correct.
4. Check it the field infos.slitdate exists
5. Check if the field infos.splitsn exists
6. Check if the field infos.splitfile exists.

Multiple datafiles

Function name: multiple_datafiles(this)
Description: Checks the function if the variable 'datafile' is of type cell (two datafiles). The markerchannel number is assigned explicitly.

Tests:

For both datafiles the same tests as in the one_datafile function are performed individually. Additionally it is tested if the number of input files does match the number of output files.

4.38 Testcases: pspm_trim

4.38.1 Information

Testclass: pspm_trim_test
Function: newdatafile=pspm_trim(datafile, from, to, reference, options)

4.38.2 Setup

If not otherwise declared, the input variable fn is referring to a datafile which was generated with pspm_testdata_gen and consists of the following channels:

data{1}.chantype = 'scr';
data{2}.chantype = 'marker';
data{3}.chantype = 'hr';
data{4}.chantype = 'hb';
data{5}.chantype = 'marker';
data{6}.chantype = 'resp';
data{7}.chantype = 'scr';

The duration of the data recording is 10s.

4.38.3 Testcases

Invalid input arguments

Function name: invalid_inputargs(testCase)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_trim(testCase.fn, [1 2], 5, ’marker’) [invalid from parameter]	ID:invalid_input
pspm_trim(testCase.fn, 0, ’bla’, ’marker’) [invalid to parameter]	ID:invalid_input
pspm_trim(testCase.fn, 0, ’[]’, ’marker’) [invalid to parameter]	ID:invalid_input
pspm_trim(fn, 0, 5) [no reference]	ID:invalid_input
pspm_trim(fn, 0, 5, 6) [no char or 2-element numeric reference]	ID:invalid_input
pspm_trim(fn, 0, 5, ‘bla’) [invalid char reference]	ID:invalid_input
pspm_trim(fn, 0, 5, [-1 5]) [invalid numeric start reference]	ID:invalid_input
pspm_trim(fn, 0, 5, [5 4]) [invalid numeric start/end reference]	ID:invalid_input

4.38.4 Reference = ‘marker’ tests

Function name: marker_tests(testCase)
Description: A wrapper function for tests with reference = ‘marker’. It executes the methods markertest_k, where the testcases are defined.

markertest_1

Description: from and to are set so that the trimming points are out of the range [0,duration]. Hence the data should not be trimmed.

Expected warning: ID: marker_out_of_range

Input: pspm_trim(fn, -20, 20, ’marker’)

markertest_2

Description: from and to are set so that the trimming points are exactly (0, duration). Hence the data should not be trimmed.

Input: from = -1 * marker(1) to = duration - marker(end) pspm_trim(fn, from, to, ’marker’)

markertest_3

Description: from and to are set so that the trimming points in the range [0,duration].

Input: pspm_trim(fn, 1, -2, ’marker’)

4.38.5 Reference = ‘file’ tests

Function name: file_tests(testCase)

Description: A wrapper function for tests with reference = ‘file’. It executes the methods filetest_k, where the testcases are defined.

filetest_1

Description: from and to are set so that the trimming points are out of the range [0,duration]. Hence the data should not be trimmed.

Expected warning: ID: marker_out_of_range

Input: pspm_trim(fn, -12.5, 50, ’marker’)

filetest_2

Description: from and to are set so that the trimming points are exactly (0, duration). Hence the data should not be trimmed.

Input: pspm_trim(fn, 0 , duration, ’marker’)

filetest_3

Description: from and to are set so that the trimming points in the range [0,duration].

Input: pspm_trim(fn,2.1, duration – 2.5, ’marker’)

Numeric reference tests

Function name: num_tests(testCase)
Description: A wrapper function for tests with reference = [a b] (a, b are two integers with a<b). It executes the methods markertest_k, where the testcases are defined.

numtest_1

Description: from and to are set so that the trimming points are out of the range [0,duration]. Hence the data should not be trimmed.

Expected warning: ID: marker_out_of_range

Input: pspm_trim(fn, -20, 20, [2 14])

numtest_2

Description: from and to are set so that the trimming points are exactly (0, duration). Hence the data should not be trimmed.

Input: from = -1 * marker(3) to = duration - marker(8) pspm_trim(fn, from, to, [3 8])

numtest_3

Description: from and to are set so that the trimming points in the range [0,duration].

Input: pspm_trim(fn, -1.5, 2, [2 7])

numtest_4

Description: Second reference point is out of the marker range; from is set to 'none'. Hence the data should not be trimmed.

Expected warning: ID: marker_out_of_range

Input: pspm_trim(fn, 'none', 0, [1 (numel(marker) + 1)])

Multiple file reference tests

Function name: multiple_files(testCase)

Description: The input variable datafile is either a cell array of two filenames or a cell array of two stucts. In both cases it is tested whether the return value is also a cell array of two filenames and whether both files are trimmed correctly.

Options tests

Marker channel number option

Function name: marker_chan_num_option_test(testCase)

Description: Tests if the option marker_chan_num is working correctly. There are two tests: Test 1: Checks for a warning if the selected channel is no marker channel. Test 2: Checks if the selected channel is actually used.

4.39 Testcases: pspm_write_channel

4.39.1 Information

Testclass: pspm_write_channel_test
Function: [sts] = pspm_write_channel(fn, newdata, action, options)

4.39.2 Setup

Testdatafile

The testdatafile is a class property. It is generated by the function generate_testdatafile() once the test class is setup. Changes made by a test to the testdatafile won't be reverted. Thus some test functions rely on the changes made by another test function. Therefore the functions may not work properly if called individually.

Structure (created with generate_testdatafile()) `data{1}.chantype = 'scr';`
```
data{2}.chantype = 'marker';
data{3}.chantype = 'scr';
```

The sampling rate is 100 Hz and the duration is 500s.

4.39.3 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
pspm_write_channel() [no parameter]	ID:invalid_input
pspm_write_channel(1) [fn is a number]	ID:invalid_input
pspm_write_channel('some_file', []) [no action passed]	ID:unknown_action
pspm_write_channel('some_file', [], ") [empty action passed]	ID:unknown_action
options.channel = 'some invalid channel' pspm_write_channel('some_file', [], 'add', options) [invalid channel]	ID:invalid_input
options.channel = -1 pspm_write_channel('some_file', [], 'add', options) [negative channel]	ID:invalid_input
options.channel = 0 pspm_write_channel('some_file', [], 'delete', options) [no channel and no data given]	ID:invalid_input
options.channel = 0 pspm_write_channel('some_file', [], 'add', options) [empty newdata]	ID:invalid_input
options.channel = 0 pspm_write_channel('some_file', 1:3, 'add', options) [newdata is not cell and not struct]	ID:invalid_input
options.channel = 1:5 pspm_write_channel(this.testdatafile, [], 'delete', options) [more given channels than in file exist]	ID:invalid_input
options.channel = 'ecg'; pspm_write_channel(this.testdatafile, [], 'delete', options)	ID:no_matching_channels
pspm_write_channel(this.testdatafile, gen_data.data{1}, 'add') [generated data has the wrong format (two rows in one channel)]	ID:invalid_data_structure

Action 'add'

Function name: test_add(this)
Description: Checks if action 'add' behaves as expected. A new channel with chantype = 'hb', sr = 200 and duration = 500 is generated.

Tests:

1. Load condition before and after and pass it to 'Verify write'

Action 'add transposed'

Function name: test_add_transposed(this)
Description: Checks if action 'add' behaves as expected, when data has the wrong dimen-

sions. A new channel with chantype = 'rs', sr = 200 and duration = 500 is generated.

Tests:

1. Transpose generated data
2. Load condition before and after and pass it to 'Verify write'

Action 'replace'/'add'

Function name: test_replace_add(this)
Description: Checks if action 'replace' behaves as expected. A new channel with chantype = 'hr', sr = 10 and duration = 500 is generated.

Tests:

1. Running pspm_write_channel with action = 'replace' should issue 'ID:no_matching_channels' (channeltype should not exist before) and then instead add the channel
2. Load condition before and after and pass it to 'Verify write'

Action 'replace'

Function name: test_replace(this)
Description: Checks if action 'replace' behaves as expected. A new channel with chantype = 'hr', sr = 20 and duration = 500 is generated.

Tests:

1. Load condition before and after and pass it to 'Verify write'
2. Test if 'hr' channel has sample rate 20

Action 'delete' (one channel)

Function name: test_delete_single(this)
Description: Checks if action 'delete' behaves as expected. In this test only one channel will be deleted. To test the delete algorithm there will be 7 channels added which are then also used for test_delete_multi(this). The particular channels are then identified by the sample rate which corresponds to the channel id * 10.

Tests:

1. Delete channel with chantype = 'hr' in newdata.header.chantype
    (a) Verify write
    (b) Ensure only one channel has been deleted
    (c) Test if there is no more channel with chantype = 'hr'

2. Delete channel with channel number in options.channel

    (a) Verify Write
    (b) Ensure only one channel has been deleted

3. Test the delete algorithm

    (a) Remove ’resp’ channel with options.delete = ’last’
        i. Verify write
        ii. Ensure only one channel has been deleted
        iii. Test if last channel was deleted
    (b) Remove ’resp’ channel with options.delete = ’first’
        i. Verify write
        ii. Ensure only one channel has been deleted
        iii. Test if last entry was not deleted

Action ’delete’ (multiple channels)

Function name: test_delete_multi(this)
Description: Checks if action ’delete’ behaves as expected. In this test only multiple channels will be deleted. This test relys on the changes made to the testdatafile by other test functions in this class.

Tests:

1. Delete channel 1 and 2 from testdatafile

    (a) Verify write
    (b) Ensure two channels have been deleted

2. Delete all ’resp’ channels from testdatafile

    (a) Verify write
    (b) Test if datafile contains no more ’resp’ channels

4.39.4 Other methods

Verify write

Is called after pspm_write_channel has been called (action = ’add’ or action = ’replace’) and tests if data was written and a new history entry was made.
Tests:

1. if action = ’add’, test if there is a new channel

2. if action = ’replace’, test if there is still the same amount of channels

3. if action = 'delete', test if there have been as many channels deleted as given in outinfos.channel
4. test if history has a new entry
5. search for channels with same chantype as added channel (should be only one channel)
6. test if number of data elements in new channel and added channel is equal
7. test if new channel and added channel have same 'sr'

5 External functions and tools

5.1 VB (Variational Bayes) inversion algorithm by Jean Daunizeau

Updated October 2014
Changes made for use in PsPM:

- *VBA_ReDisplay.m*, fixed try-catch syntax in various places by adding a comma after "try" to avoid warning in matlab > 2007
- *VBA_inv.m*, line 42: added warning off/on to suppress the warning "Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND = NaN."

Updated October 2016
Changes made for use in PsPM:

- *VBA_ReDisplay.m*, fixed try-catch syntax in various places by adding a comma after "try" to avoid warning in matlab > 2007
- *VBA_inv.m*, line 48: added warning off/on to suppress the warning "Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND = NaN."
- *VBA_NLStateSpaceModel.m*: added resetting warning to preceeding state.

6 List of functions

Name	Main author	Test exists	Test Doc
f_SCR	Dominik Bach & Jean Daunizeau	-	-
f_SF	Dominik Bach	-	-
g_SCR	Dominik Bach	-	-
pspm	Dominik Bach	x	x
scr	Dominik Bach	x	x
pspm_align_channels	Dominik Bach	x	x
pspm_axpos	Dominik Bach	-	-
pspm_bf_brf	Saurabh Khemka & Dominik Bach	-	-

pspm_bf_data	Dominik Bach	x	x
pspm_bf_FIR	Dominik Bach	-	-
pspm_bf_Fourier	Dominik Bach	-	-
pspm_bf_hprf	Dominik Bach	-	-
pspm_bf_hprf_e	Tobias Moser	-	-
pspm_bf_hprf_fc	Tobias Moser	-	-
pspm_bf_hprf_fc_f	Tobias Moser	-	-
pspm_bf_lcrf_gm	Tobias Moser	-	-
pspm_bf_ldrf_gm	Tobias Moser	-	-
pspm_bf_ldrf_gu	Tobias Moser	-	-
pspm_bf_psrf_fc	Tobias Moser	-	-
pspm_rarf_e	Tobias Moser	-	-
pspm_rarf_fc	Tobias Moser	-	-
pspm_rfrrf_e	Tobias Moser	-	-
pspm_rprf_e	Tobias Moser	-	-
pspm_bf_scrf_f	Dominik Bach	-	-
pspm_bf_scrf	Dominik Bach	-	-
pspm_bf_spsrf_box	Laure Ciernik	-	-
pspm_bf_spsrf_gamma	Laure Ciernik	-	-
pspm_butter	Dominik Bach	X	X
pspm_compute_visual_angle	Laure Ciernik	-	-
pspm_con1	Dominik Bach	-	-
pspm_con2	Dominik Bach	-	-
pspm_contrast	Dominik Bach	-	-
pspm_convert_area2diameter	Tobias Moser	-	-
pspm_convert_au2mm	Tobias Moser	-	-
pspm_convert_illum2lum	Tobias Moser	-	-
pspm_convert_lux2cdm2	Tobias Moser	-	-
pspm_convert_mm2visdeg	Tobias Moser	-	-
pspm_convert_pixel2unit	Laure Ciernik	-	-
pspm_convert_unit	Tobias Moser	x	x
pspm_convert_visangle2sps	Laure Ciernik	-	-
pspm_data_editor	Tobias Moser	-	-
pspm_dcm_inv	Dominik Bach	-	-
pspm_dcm	Dominik Bach	-	-
pspm_denoise_spike	Dominik Bach	-	-
pspm_display	Philipp C Paulus	-	-
pspm_down	Dominik Bach	x	-
pspm_downsample	Dominik Bach	-	-
pspm_ecg2hb	Philipp C Paulus	x	x
pspm_ecg2hb_amri	Eshref Yozdemir	x	-
pspm_ecg_editor	Tobias Moser	-	-
pspm_exp	Dominik Bach	x	-

pspm_extract_segments	Tobias Moser	x	x
pspm_filtfilt	Dominik Bach	x	x
pspm_find_channel	Dominik Bach	x	x
pspm_find_data_epochs	Tobias Moser	-	-
pspm_find_sounds	Samuel Gerster	x	x
pspm_find_valid_fixations	Tobias Moser	x	x
pspm_get_acq_bioread	Tobias Moser	x	x
pspm_get_acq	Dominik Bach	x	x
pspm_get_acqmat	Dominik Bach	x	x
pspm_get_biograph	Dominik Bach	x	x
pspm_get_biosemi	Dominik Bach	x	x
pspm_get_biotrace	Dominik Bach	x	x
pspm_get_blink_l	Tobias Moser	-	-
pspm_get_blink_r	Tobias Moser	-	-
pspm_get_brainvis	Dominik Bach	x	x
pspm_get_cell	Dominik Bach	-	-
pspm_get_cnt	Dominik Bach	-	-
pspm_get_custom	Tobias Moser	-	-
pspm_get_ecg	Dominik Bach	x	x
pspm_get_edf	Tobias Moser	x	x
pspm_get_events	Dominik Bach	x	x
pspm_get_eyelink	Christoph Korn, Tobias Moser	x	x
pspm_get_gaze_x_l	Tobias Moser	-	-
pspm_get_gaze_y_l	Tobias Moser	-	-
pspm_get_gaze_x_r	Tobias Moser	-	-
pspm_get_gaze_y_r	Tobias Moser	-	-
pspm_get_hb	Dominik Bach	x	x
pspm_get_hp	Dominik Bach	-	-
pspm_get_hr	Dominik Bach	x	x
pspm_get_labchartmat_ext	Dominik Bach	x	x
pspm_get_labchartmat_in	Dominik Bach	x	x
pspm_get_marker	Dominik Bach	x	x
pspm_get_markerinfo	Dominik Bach	-	-
pspm_get_mat	Dominik Bach	x	x
pspm_get_obs	Linus RÃŒttimann	x	x
pspm_get_physlog	Tobias Moser	-	-
pspm_get_pupil	Dominik Bach	x	x
pspm_get_pupil_l	Tobias Moser	-	-
pspm_get_pupil_r	Tobias Moser	-	-
pspm_get_resp	Dominik Bach	x	x
pspm_get_rf	Dominik Bach	-	-
pspm_get_saccade_l	Laure Ciernik	-	-

pspm_get_saccade_r	Laure Ciernik	-	-
pspm_get_scr	Dominik Bach	x	x
pspm_get_smi	Eshref Yozdemir	-	-
pspm_get_sps	Laure Ciernik	-	-
pspm_get_spike	Dominik Bach	x	x
pspm_get_sound	Tobias Moser	-	-
pspm_get_timing	Dominik Bach	x	x
pspm_get_txt	Dominik Bach	x	x
pspm_get_vario	Dominik Bach	x	x
pspm_get_viewpoint	Eshref Yozdemir	-	-
pspm_get_wdq	Dominik Bach	-	-
pspm_get_wdq_n	Tobias Moser	x	x
pspm_glm_recon	Dominik Bach	-	-
pspm_glm	Dominik Bach	x	x
pspm_hb2hp	Dominik Bach	x	x
pspm_hb2hr	Dominik Bach	-	-
pspm_import	Dominik Bach	x	x
pspm_init	Dominik Bach	-	-
pspm_interpolate	Tobias Moser	x	x
pspm_jobman	Gabriel GrÃ€ni	-	-
pspm_job_create	Dominik Bach	-	-
pspm_load_data	Dominik Bach	x	x
pspm_load1	Dominik Bach	x	x
pspm_load_single_chan	Eshref Yozdemir	-	-
pspm_merge	Dominik Bach	-	-
pspm_path	Eshref Yozdemir	x	-
pspm_peakscore	Dominik Bach	-	-
pspm_pp	Dominik Bach	x	x
pspm_ppu2hb	Samuel Gerster	-	-
pspm_predval	Dominik Bach	-	-
pspm_prepdata	Dominik Bach	x	x
pspm_process_illuminance	Tobias Moser	x	-
pspm_pulse_convert	Dominik Bach	x	-
pspm_pupil_correct_eyelink	Eshref Yozdemir	x	-
pspm_pupil_correct	Eshref Yozdemir	x	-
pspm_pupil_pp	Eshref Yozdemir	x	-
pspm_pupil_pp_options	Eshref Yozdemir	-	-
pspm_quit	Dominik Bach	-	-
pspm_ren	Dominik Bach	x	x
pspm_resp_pp	Dominik Bach	-	-
pspm_rev_con	Dominik Bach	-	-
pspm_rev_dcm	Dominik Bach	-	-
pspm_rev_glm	Dominik Bach	-	-

pspm_rev2	Dominik Bach	-	-
pspm_review	Gabriel Graeni	-	-
pspm_segment_mean	Tobias Moser	-	-
pspm_scr_pp	Dadi Zhao	x	x
pspm_sf_auc	Dominik Bach	-	-
pspm_sf_dcm	Dominik Bach	-	-
pspm_sf_mp	Dominik Bach	-	-
pspm_sf_scl	Dominik Bach	-	-
pspm_sf_theta	Dominik Bach	-	-
pspm_sf	Dominik Bach	-	-
pspm_sf_get_theta	Dominik Bach	-	-
pspm_show_arms	Dominik Bach	-	-
pspm_spike_convert	Dominik Bach	-	-
pspm_split_sessions	Linus RÃŒttimann	x	x
pspm_transfer_function	Dominik Bach	-	-
pspm_trim	Dominik Bach	x	x
pspm_ui	Dadi Zhao	x	x
pspm_version	Tobias Moser	-	-
pspm_write_channel	Tobias Moser	x	x
set_blinks_saccades_to_nan	Eshref Yozdemir	x	-