Subject Array and Wrapper Automation (SAWA) - version 1 -

SAWA is a toolbox that automatically wraps any command line, matlab, or batch function using subject-specific or iterative variable inputs.

Contents:

Highly Suggested:
-SPM (for batch editor and spm_select): http://www.fil.ion.ucl.ac.uk/spm/software/

-xlwrite (for mac users): http://www.mathworks.com/matlabcentral/fileexchange/38591-xlwrite--generate-xls-x--files-without-excel-on-mac-linux-win

-findjobj (for horizontal scrollbar in notes):
http://www.mathworks.com/matlabcentral/fileexchange/14317-findjobj-find-java-handles-of-matlab-graphic-objects

Citation:

Theiss, J.D. (2015). Subject Array and Wrapper Automation (SAWA) [Computer software]. https://github.com/jdtheiss/sawa.

Introduction

The main component of SAWA is the subject array, a structural array that can contain subject-specific information to be utilized in wrapping functions. At its simplest, SAWA is an organizational tool that can maintain up-to-date information as well as record analyses and other notes. However, SAWA is built to feed information from the subject array to wrappers for any batch, function, or command. As such, SAWA provides users the ability to perform complex analyses using subject data in SPM, AFNI, FSL, etc., or any unix/C/matlab commands.

With a wrapper editor for batch utility, command line, and matlab functions, users can build simple pipelines for repeat analyses or create their own programs using their own functions. The batch editor directly uses SPM's batch utility system, which allows users to directly choose variables that will be filled by the subject array or other functions/variables. The command line editor allows users to wrap command line functions while also selecting command line switches to use and set. Finally, the function wrapper utility provides users the ability to wrap matlab functions by setting input arguments and selecting output arguments to be returned.

As a way to record analyses that users have run, SAWA also prints inputs/outputs/errors. Users can add notes into the analysis records and save records for different subject arrays (e.g. different studies). Records are stored by analysis name and date.

Installation

SAWA is available for download and is updated on github (see citation above for link). After you have downloaded the zipped file folder, go into the "sawa/main/functions" folder and open "sawa.m" in matlab. Simply run the function, and SAWA will be automatically added to your matlab path. At this point, you may close the "sawa.m" file and begin using SAWA through the graphical user interface (GUI).

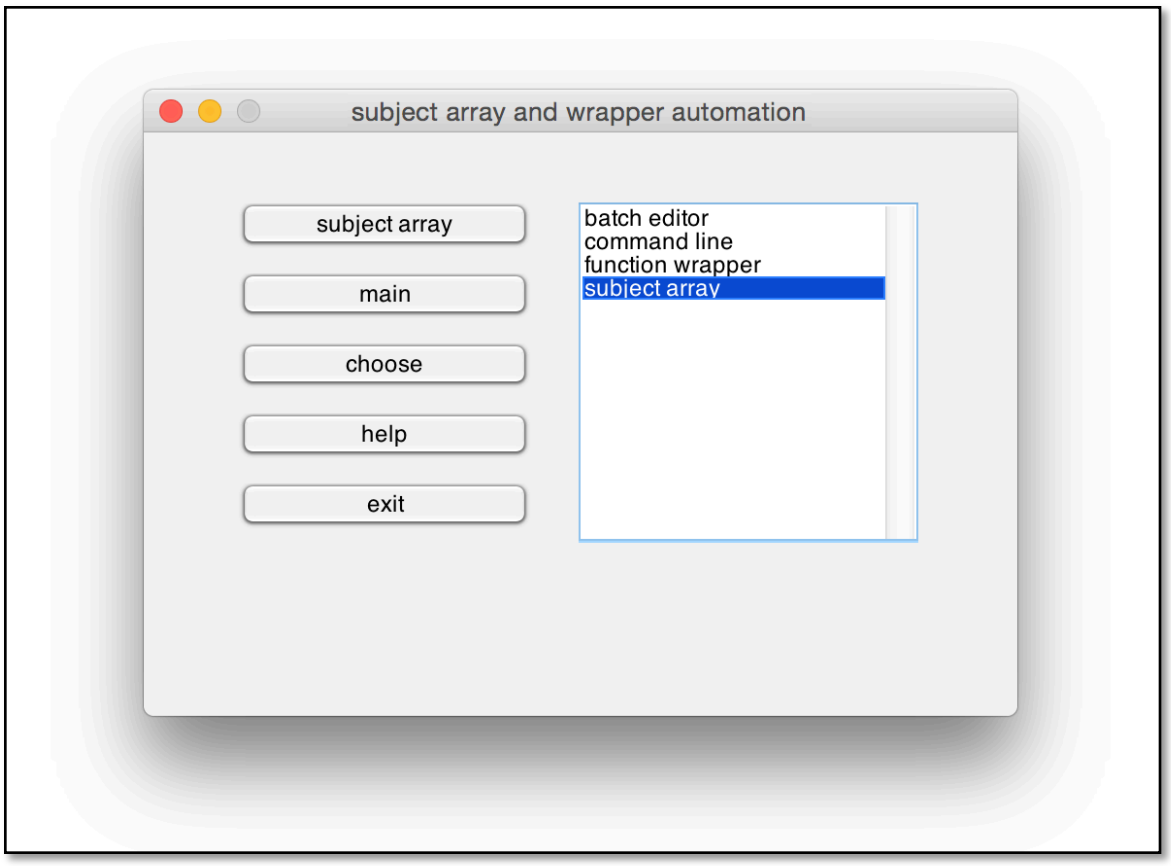


Figure 1. Main GUI page for SAWA includes folders in “sawa” folder.

Getting Started

Once you have installed and run the “sawa.m” file, you may begin using any of the utilities that SAWA provides. SAWA uses a graphical user interface (GUI) for simple point-and-click usage, but each SAWA function can also be run from the command prompt as well as within other functions. By typing sawa into the matlab command prompt, the main GUI opens (Figure 1).

The list on the right side of the GUI displays the folders within the “sawa” folder. Users may then add other folders/functions to the same directory if they wish to use them in SAWA. The “subject array” button allows users to choose a current subject array (a “subjects.mat” file that contains subject arrays). Initially, there will be no subject array to choose. The “main” button returns the user to the main folders whereas “choose” will reveal the contents of the selected folder in the list. After choosing a folder, the “choose” button will change to “add”. Clicking “add” will load the function into the queue of functions to be run. When a function is selected in the list, pressing “help” on the main GUI page will display the function’s help message in the command prompt. Finally, the “exit” button will quit SAWA.

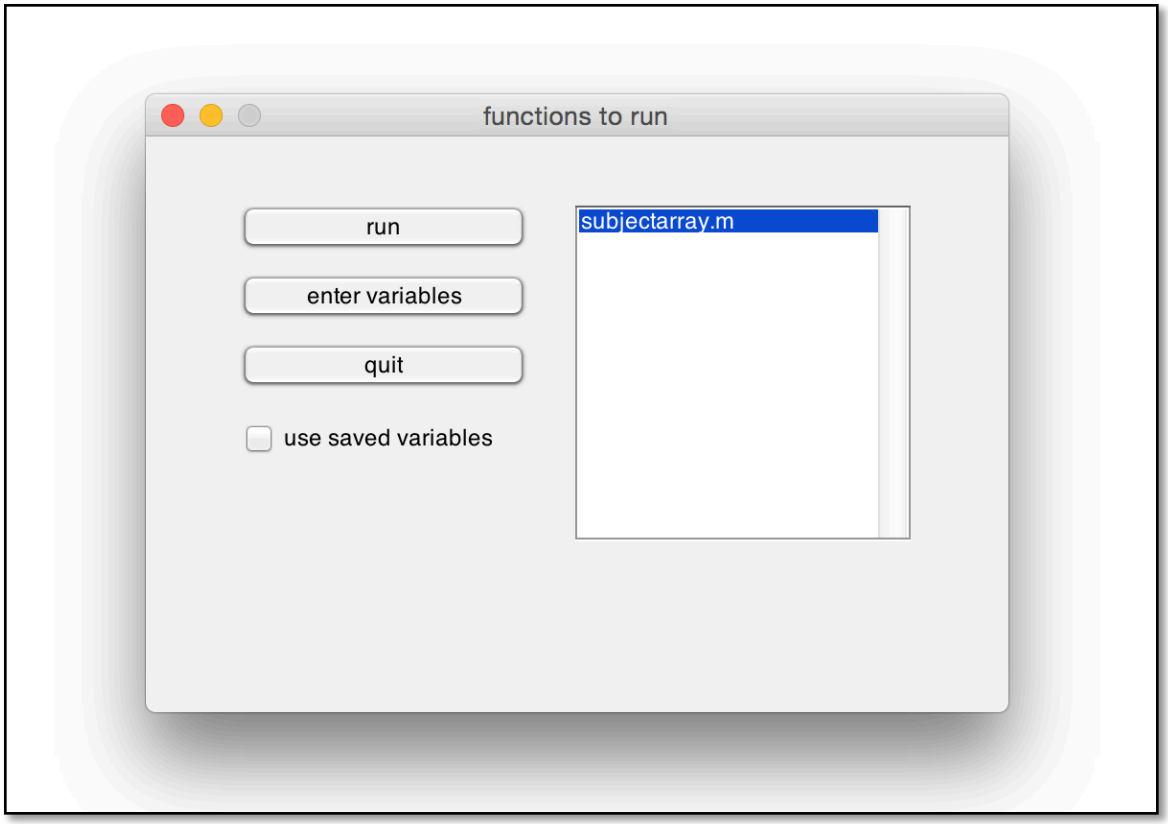


Figure 2. Functions to run GUI page for SAWA includes any files within the previously selected folder.

After choosing a function to "add", the user is brought to the "functions to run" GUI page. The "functions to run" GUI page contains the list of functions that will be run, and new functions can by selecting them from the main GUI page and pressing "add". Pressing "run" will run each function in order. For the wrapper editors (i.e. batch editor, command line, and function wrapper), this will not automatically run since the user needs to input functions/variables to use. However, for functions that do not require variable input (i.e. a matlab script/function), the function will run immediately. The "enter variables" button allows users to enter the variables to input for each function prior to running. Once pressed, each function in the queue that can have its variables set will open for the user to input variables. This is useful for lengthy functions to avoid waiting for one function to finish before setting the next set of variables. The "use saved variables" checkbox allows users to choose previously saved variables (as "*_savedvars.mat") with the function. This is also how the "enter variables" button works in that each set of variables is saved to the "sawa/main/jobs" folder and automatically retrieved. Saved variables can also be set in the wrapper editors by selecting "save" after entering the desired functions and variables. Finally, the "quit" button will clear the queue of functions to run and close the "functions to run" GUI page.

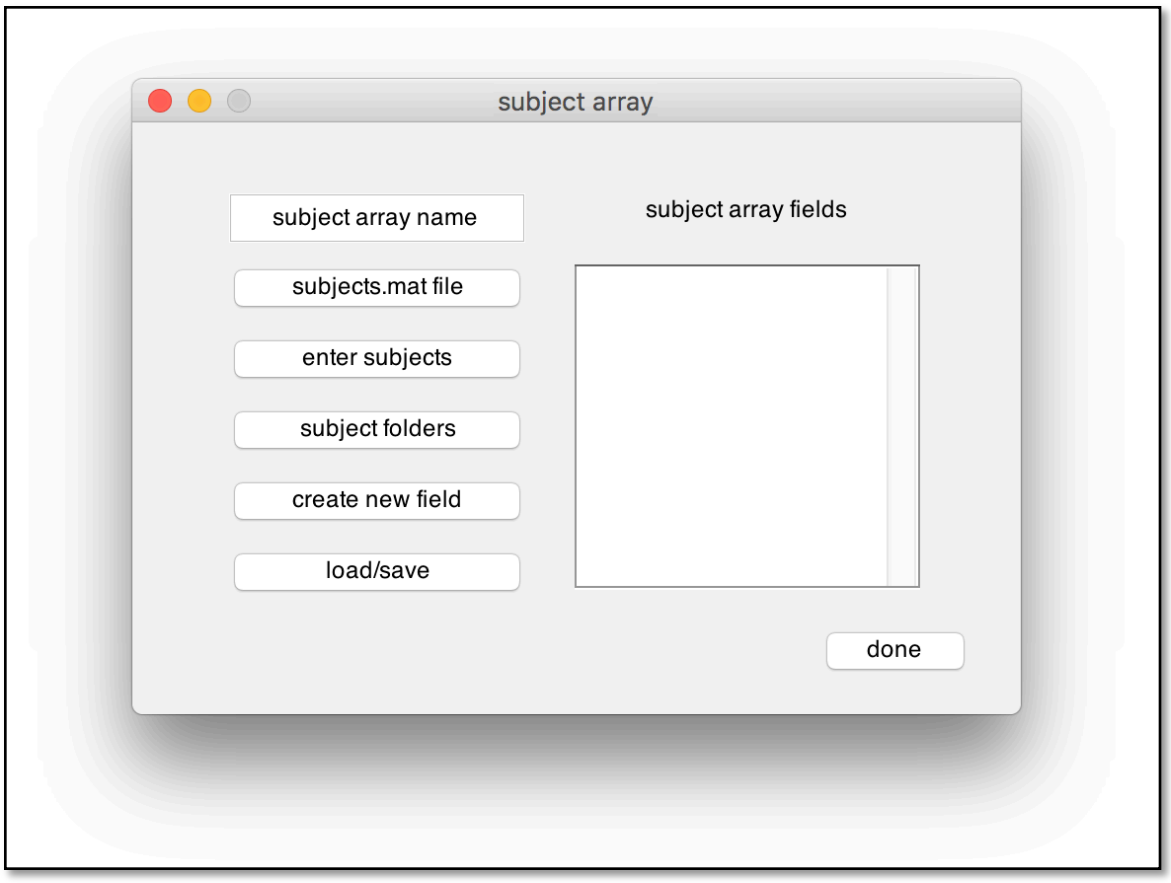
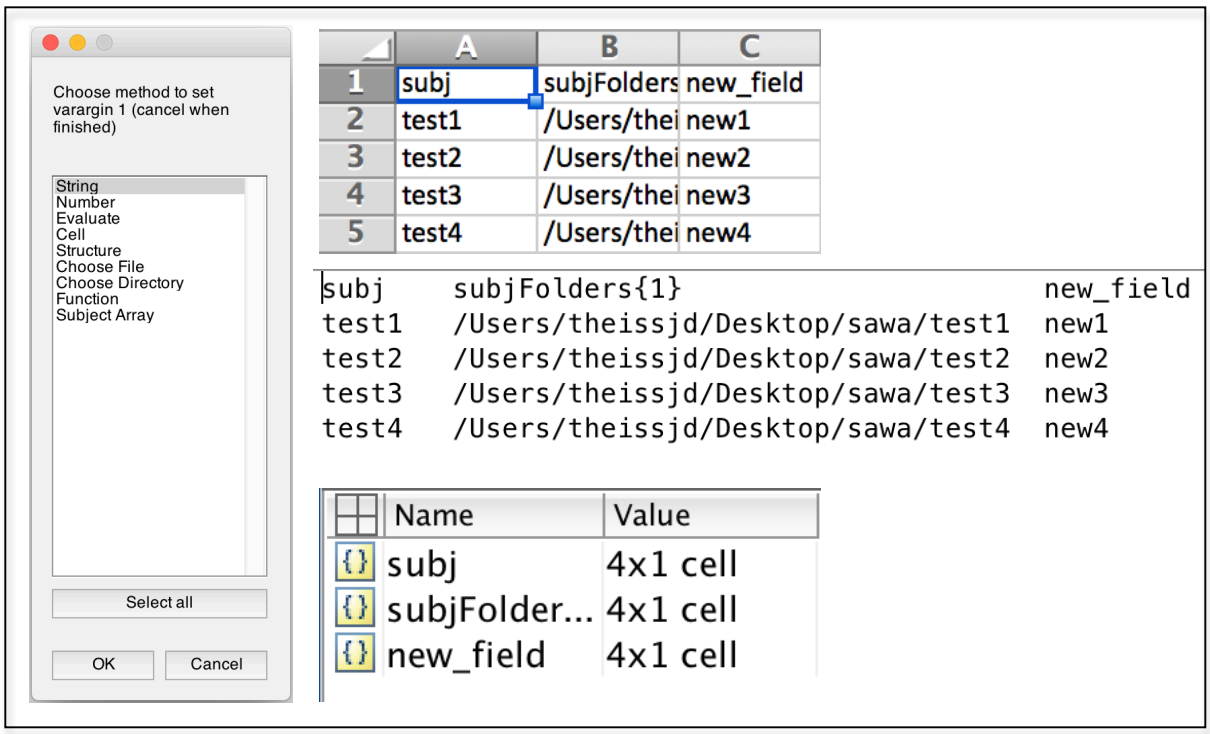


Figure 3. The subject array GUI allows users to create/load the subject array.

Creating the Subject Array

Running the "subjectarray" function within the "subject array" folder, users are able to create or load a subject array (Figure 3). The subject array is a simple structural array for subject-specific information that is then used with the wrapper editors and other functions. Each subject array should have at least a "subj" and "subjFolders" field. The "subj" field contains the subject ID as a character array (e.g., "Subject001"), and the "subjFolders" field contains the main folder location for each subject as a cellstring array (e.g., {"/Volumes/Subj001"}). Users may then add any other subject fields as they wish.

The "Subject Array Name" text box allows users to name the subject array (e.g., "ptsd" or "gonogo"). The subject array name is used to save the subject array in a "subjects.mat" file and is displayed when selecting a subject array to use. The "subjects.mat" file is simply the file that contains the subject array(s). As such, a user may choose to save a subject array to a previously created "subjects.mat" file or select a folder to save a new "subjects.mat" file. This is accomplished by pressing the "subjects.mat file" button. The "enter subjects" buttons allows users to add subjects to the subject array (and create the "subj" field). Pressing "subject folders" provides users the option to either choose a main folder that will be appended with each subject's subject ID or set manually. If choosing a main folder, the user can also choose to create each subject's main folder within the directory.



a. b.

Figure 4. The methods to set variables (a). Methods to save/load subject arrays: excel (b, top), text (b, middle), or .mat file (b, bottom).

The "create new field" button allows users to enter a fieldname to add to the subject array and choose the values to be set for chosen subjects. Setting variables to a field involves choosing a method by which to set the variables (Figure 4a). These methods are also the way in which users set variables for the wrapper editors as well. The first three methods (String, Number, and Evaluate) all involve an input box. Each row of the input box corresponds to a different subject. Furthermore, the method corresponds to a different type of value. String values are made of characters (e.g., "Subject001"); number values are numeric (e.g., 12), and evaluated values are evaluated by matlab (e.g., {12} or strrep("Subject001","001","002")). The "Cell" method will allow users to create cell variables. The "Structure" method allows users to create a structure array within the field (e.g., "Responses.Values"). When this method is selected, the user enters subfield(s) to create within the initial field. Then with each subfield the user will choose a method to set the value within the subfield. The "Choose File" and "Choose Directory" methods will open a file/directory selection window (the same as SPM uses). The "Function" method allows users to create values based on the outputs of a matlab function. For example, the user could use the "strcat" function with inputs of "Subject" and "001", "002", "003" (one per row) to create "Subject001", "Subject002", "Subject003". Once a subject array is created, an additional method called "Subject Array" is made available. With this method, users are able to select subject-specific variables to use to be set to the subject array. For example, one could easily set a "RestingState" field by selecting the "subjFolders" field then entering "/RestingState" as a subfolder.

To simplify the way in which a subject array is created, the “load/save” button can be used to load a subject array from any of the following files: excel, text, or .mat file. Additionally, a subject array can be loaded from a previously created “subjects.mat” file. For each file type, the way in which fields are set up is very similar. For excel files, the headers correspond to the fieldnames (Figure 4b, top). As such, if a field has a subfield/cell, it should be included in the name (e.g., “subjFolders{1}” no “subjFolders”). Each row below the headers then corresponds to a different subject. Similarly, a text file can be used as well (Figure 4b, middle). The same procedure is followed for text files and each field should be separated by a tab. Finally, a subject array can also be loaded from a .mat file with the fields of the .mat file corresponding to the fieldnames (Figure 4b, bottom). However, since .mat file fields cannot contain special characters (e.g., {1} or .subfield), the field should only contain the fieldname with the appropriate subfields reflected in the variable itself. For example, the “subjFolders” field would contain inner cells for each subject as below.

subjFolders = {{“/Volumes/Subj001”};{“/Volumes/Subj002”};{“/Volumes/Subj003”}}

In addition to loading subject arrays, subject arrays can be written to excel/text/.mat files to continually update information in the subject array or the file. Once a subject array has been set up, simply choose “save” and then choose the type of file(s) to save the subject array to. At this point, the user can also create the “subjects.mat” file with the subject array. The location of the “subjects.mat” file will then be used for any further reference for the subject array.

Finally, users can edit/copy/delete subject array fields by clicking on the fieldname in the list on the right side of the subject array GUI. Note that deleting subject array fields will not remove the field from the subject array unless all subjects have empty field values. Otherwise, deleting for selected subjects will simply empty the field.

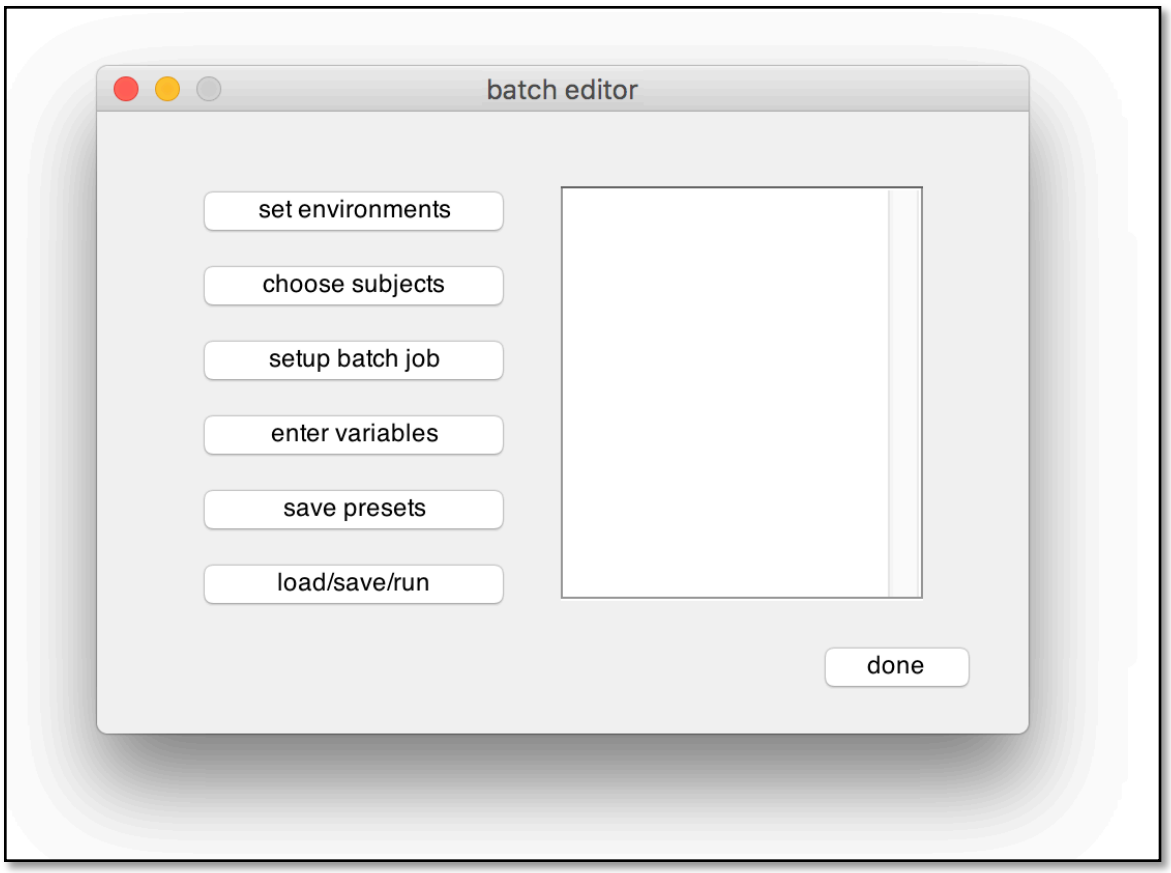


Figure 5. GUI for batch editor is similar to GUIs for other wrapper editors and pipelines.

Wrapper Editors

Each GUI for the wrapper editors (i.e. batch editor, command line, and function wrapper) has the same layout (Figure 5). The same GUI is also loaded for any pipeline that is saved. The right side of the GUI will contain the module/function names to be run. When module/function names are listed, the user may right-click on the name and copy/delete the module/function from the list.

The first button, “set environments”, is used for setting environments/paths for external programs/functions (e.g., FSL). Certain functions/programs, such as command line functions, have specific environments that need to be set in order to be run in matlab. This is also the way in which a user can add the path to a matlab function as well. If a set of functions is saved as a pipeline, the environments will automatically be set each time it is run.

The “choose subjects” button allows users to select subjects to include and whether to run the functions as “per subject” or “per iteration”. If iterations are chosen, the user will be asked to input the number of iterations. Note that if the “choose subjects” button is not pressed, functions will automatically be assumed to be “per iterations” and the number of iterations will be matched to the number of variables selected.

The “setup batch job” and “enter variables” buttons will vary based on the wrapper editor, but all serve a similar purpose. The “setup batch job” (or “add function”) button is used to choose/load functions into the editor and choose the variables to be filled out by SAWA. See each wrapper editor for specifics regarding these buttons.

After functions/variables have been chosen, users can choose to save the preset functions and variables as a pipeline for later use. Pipelines are treated similarly to the wrapper editors in that the pipeline of functions will be automatically loaded into the GUI each time the user runs it. This is extremely helpful for analyses/steps that will be performed several times. For example, a pipeline can be created for SPM fMRI preprocessing or FSL’s DTI analysis. The variables chosen can then be set each time the user runs the pipeline without having to reload the functions.

Finally, users can load, save, or run the pipeline of functions by clicking “load/save/run”. Users may load previously saved variables (saved “sawa/main/jobs/*_savedvars.mat”) and then edit the variables in the GUI. Similarly, the variables can be saved into the “jobs” folder for later use. This is helpful for specific jobs that do not necessarily need to be pipelines. For example, running several two-sample t-tests for SPM is easiest by reloading the previous saved variables and changing the contrast image, etc. Clicking “run” will automatically begin running the functions with the chosen variables. After the functions have completed an “output” variable will be created in the base workspace that will contain the outputs per iteration and function.

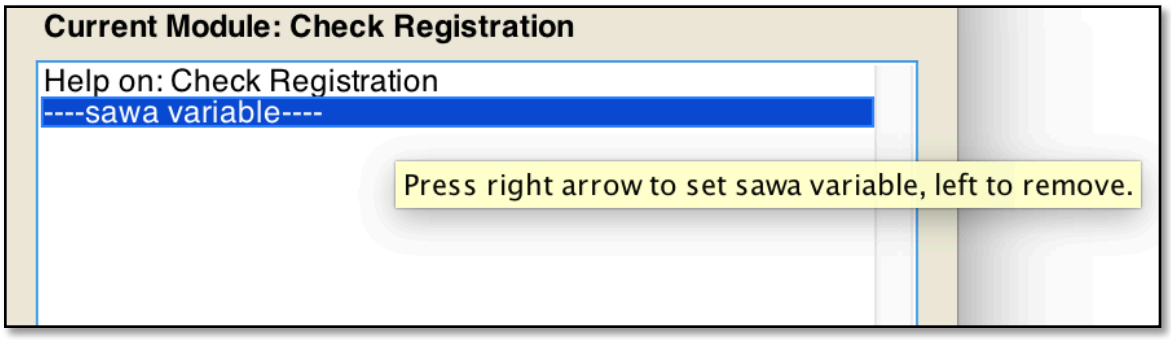


Figure 6. Selecting variables to be filled out by SAWA in batch editor.

Using the Batch Editor Utility

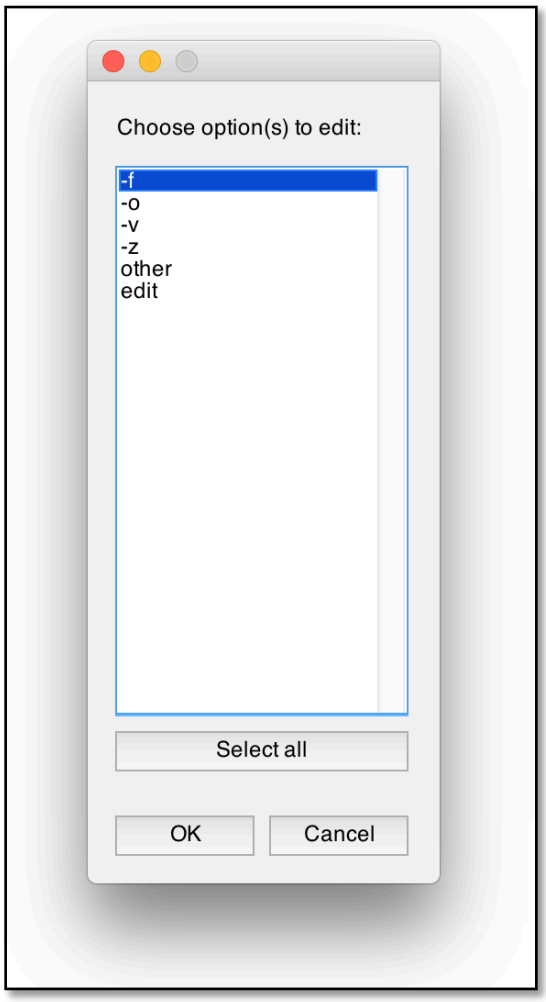
With the batch editor utility, users can create SPM batches that pull information from the subject array and/or other functions/variables to complete jobs. Simply click “setup batch job” to load the matlab batch editor, then load or choose functions to use. Next, within the batch editor, choose variables to be filled out with SAWA by pressing the right arrow on the keyboard when the item is highlighted (Figure 6). Similarly, to remove a variable, press the left arrow on the keyboard with the item highlighted. After choosing variables to be filled out with SAWA, enter any other variables directly into the batch editor and close the editor. Finally, choose a folder (either through the file system or subject array) to save the JOBS/BatchName/BatchFile.mat and choose whether to run or save the jobs. Also, if the batch uses SPM directly, the user will be asked whether to overwrite previous

files (i.e. SPM.mat files). Choosing to overwrite SPM files will automatically bypass the warning dialogs displayed when such an instance occurs.

After choosing subjects/iterations, click on the module to edit and then click "enter variables". This will bring up the variables to be set for the chosen module. For each item, the user will be asked how they would like to set the variable (see Figure 4a and page 6 for overview).

Using the Command Line Utility

With the command line utility, users can easily set and wrap command line functions. Depending on the program being called, users will likely need to set the matlab path to the directory of the desired functions. Simply click "set environments" and enter e.g. PATH and choose the folder containing the functions. (Note: if presets are saved, environments will automatically be set when using the pipeline).



Next, enter the functions to use. For each function, SAWA will attempt to show the help information and extract command line switches (Figure 7). Set the command line switches in the same manner as setting any other variables. As with any of the wrapper editors, pipelines can be saved for the command line utility, and "savedvars.mat" files can be loaded or saved as well. The printout for command line utility pipelines will be the command (i.e. function –options variables) and the command line output that is shown in the command window.

Figure 7. Command line switches are automatically extracted from help information.

Using the Wrap Functions Utility

The wrap functions utility is very similar to the command line utility, with the only difference being the source of the commands. This utility allows users to wrap almost any matlab function. Note that these should be functions (i.e. with "function" at the top of the script in matlab editor). As with the command line utility, users may want to set certain script locations to the matlab path using the "set environments" button. Then, simply enter the functions to be called, set the input arguments and select output arguments.

Appendix A. Functions

any2str.m

out = any2str(maxrow,varargin)
This function will convert any class to its string representation.

Inputs:
maxrow - max row that an input can have (see example). Default is inf.
varargin - any class to be input (see example)

Outputs:
out - string representation equal to the number of arguments input

Example:
```
out = any2str(2,'test',{'test';10},[1,2,3],@disp,{struct('testing',{'this'}),12,'testing'})
out =

  'test'  'test'  [1 2 3]   @disp    {[1x1 struct] 12 'testing'}
         10
```

Note: for vertical cell arrays, no '{}'s are added (see example above).

Created by Justin Theiss

auto_batch.m

varargout = auto_batch(cmd,varargin)
This function will allow you to set arguments for the command line function funname and then run the function.

Inputs:
cmd - command to use (i.e. 'add_funciton','set_options','auto_run')
varargin - arguments to be passed to cmd

Outputs:
fp - funpass struct containing variables from call cmd(varargin)

Example:
```
funcs = matlabbatch;
options{1,1}(1:4,1) = {'test1','test2','test3','test4'};
itemidx{1}(1) = 3;
rep{1}(1) = 0;
funrun = 1:4;
fp = struct('funcs',{funcs},'options',{options},'itemidx',{itemidx},'rep',{rep},...
'funrun',{funrun});
fp = auto_batch('run_batch',fp)
```

requires: choose_fields choose_spm closedlg funpass printres
sawa_createvars sawa_evalchar sawa_evalvars sawa_find
sawa_savebatchjob sawa_setbatch sawa_setdeps sawa_setfield
sawa_setupjob sawa_strjoin settimeleft subidx

Created by Justin Theiss

auto_cmd.m

varargout = auto_cmd(cmd,varargin)
This function will allow you to set arguments for the command line function
funname and then run the function.

Inputs:
cmd - command to use (i.e. 'add_funciton','set_options','auto_run')
varargin - arguments to be passed to cmd

Outputs:
fp - funpass struct containing variables from call cmd(varargin)
- output - the char output from the command line (including the command
and options used, see example)
- funcs - cell array of functions used
- options - cell array of options used
- subrun - numeric array of subjects/iterations run

Example:
fp = struct('funcs',{'echo'},'options',{' this is a test'});
fp = auto_cmd('auto_run',fp)
Command Prompt:
1
echo this is a test
this is a test

fp =
    funcs: 'echo'
  options: ' this is a test'
   output: {{1x1 cell}}

fp.output{1}{1} =

this is a test

requires: funpass printres sawa_cat sawa_createvars sawa_evalvars
sawa_setfield sawa_strjoin sawa_system settimeleft

Created by Justin Theiss

auto_wrap.m

```
varargout = auto_wrap(cmd,varargin)
This function will automatically create a wrapper to be used with chosen
function and subjects.

Inputs:
cmd - command to use (i.e. 'add_funciton','set_options','run_cmd')
varargin - arguments to be passed to cmd

Outputs:
fp - funpass struct containing variables from call cmd(varargin)
- output - the chosen output from the function
- funcs - cell array of functions used
- options - cell array of options used
- subrun - numeric array of subjects/iterations run

Example:
funcs = 'strrep'
options(1,1:3) = {{'test'},{'e'},{'oa'}};
fp = struct('funcs',{funcs},'options',{options})
auto_wrap('auto_run',fp)
Command Prompt:
1
strrep(test, e, oa)
varargout
toast

fp =
  funcs: 'strrep'
options: {{1x1 cell} {1x1 cell} {1x1 cell}}
 output: {{1x1 cell}}

fp.output{1}{1} =

toast

requires: any2str cell2strtable funpass getargs printres sawa_cat
sawa_createvars sawa_evalvars sawa_setfield sawa_strjoin settimeleft

Created by Justin Theiss
```

cell2strtable.m

```
strtable = cell2strtable(celltable,delim)
Create string table from a cell table (different from matlab's table)
with a specified delimiter separating columns.
```

Input:
-celltable - a cell array with rows and columns to create string table
-delim - (optional) delimiter to separate columns (defualt is tab)

Output:
-strtable - a string table based on the celltable with equally spaced
columns based on delimeter

Example:
celltable = [{'Column 1 Title','Column 2 Title',''};...
{'Row 2 Column 1 is longer...','Row 2 Column 2','Extra Column!'}];
delim = '\t';
strtable = cell2strtable(celltable,delim);
strtable =
Column 1 Title Column 2 Title
Row 2 Column 1 is longer... Row 2 Column 2 Extra Column!

Note: due to differing fonts, this works best with "Courier New"
(i.e. matlab command prompt font).

Created by Justin Theiss

choose_SubjectArray.m
varargout = SubjectArray(fileName,task)
holder for subjects.mat location

Inputs:
fileName - fullpath to the subjects.mat file
task - string representing task to load

Outputs:
if no inputs, fileName will be returned
if empty fileName is input, fileName file will be chosen and returned
if fileName only is input, task will be chosen and fileName and task will be returned
if fileName and task input, task will be loaded and subject array and task will be returned

Created by Justin Theiss

choose_fields.m
flds = choose_fields(sa, subrun, msg)
Choose string represntations of fields from subject array

Inputs:
sa - subject array (default will have user choose subject array)
subrun - subjects to choose fields from (default is all indices of sa)

msg - message (default is 'Choose fields:')

Outputs:
flds - cellstr of string representations of subject array field choices
(see example)

Example:
```
sa = struct('age',{{10,12},{11,13},{8,10},{11,15}});
subrun = 1:4;
msg = 'Choose subject array field to use:';
flds = choose_fields(sa,subrun,msg);
[chose 'age' field]
[chose indices '1' and '2']
flds =
  'age{1}'   'age{2}'
```

requires: sawa_subrun

Created by Justin Theiss

choose_spm.m

choose_spm
This function will allow choosing between multiple SPM versions

Inputs:
spmver - (optional) string spm version to set

Outputs:
spmver - spmver in use (or empty if failed)

example:
```
spmver = choose_spm('spm12');
```

Created by Justin Theiss

closedlg.m

closedlgs(figprops,btnprops)
This function will set a timer object to wait until a certain
dialog/message/object is found using findobj and will then perform a
callback function based on button chosen.

Inputs:
-figprops -(optional) object properties that would be used with findobj
(e.g., a cellstr like {'-regexp','name','SPM.*'}), a string to be
evaluated (e.g., 'findobj'), or a figure handle. Default is 'findobj'
-btnprops -(optional) button properties that would be used with findobj,

an index of button (for questdlg) to be pressed, or the button handle.
Default is 0 which gets the CloseRequestFcn.
-timeropts -(optional) timer options to set (see timer help doc) in a
cellstr (e.g., {'TasksToExecute',Inf,'Period',0.001})

Example:
closedlg('findobj',{'string','stop'});
cfg_util('runserial',matlabbatch);

The above example would create a timer that would press the stop button
if a dialog box such as SPM's overwrite warning were to appear.

NOTE: The default taskstoexecute is 10000 (which is approx 10000 seconds)
and the default stopfcn is @(x,y)stop_timer(x) which will delete the
timer after the taskstoexecute.

Created by Justin Theiss

common_str.m
str = commmon_str(strs)

This function will find the greatest common string (str) among a cell
array of strings (strs)

Inputs:
- strs - cell or character array of strings

Outputs:
- str - longest common string among strs

Example:
strs = {'TestingString123','ThisString2','DifferentStrings'}
str = common_str(strs)
str = 'String';

Created by Justin Theiss

convert_paths.m
sa = convert_paths(sa,task,fileName)
This function will convert the paths in the subject array sa from Mac to
PC or vice versa.

Inputs:
sa- a subject array (or any other array that holds file paths)
task- (optional) the name of the subject array
fileName- (optional) the file name of the .mat file where the subject

newpaths- (optional) cellstr of new file paths (if known, BE CAREFUL)

Outputs
sa - converted array

Example:
sa = gng;
task = 'gng';
fileName = '/Applications/sawa/Subjects/subjects.mat';
newpaths = '/Volumes/J_Drive';
sa = convert_paths(sa,task,fileName,newpaths);
This will replace the filepath J:\ with /Volumes/J_Drive and switch file separators from \ to /.

requires: match_string sawa_find subidx

Created by Justin Theiss

funpass.m

funpass(structure,vars)
This function allows you to pass variable structures between functions easily. If only one input argument is entered, structure will be returned as well as variables created from the fields in structure. If two input arguments are entered, the structure is updated using the evaluated variables in vars.

Inputs:
-structure - structure array with fields that are the intended variables to use
-vars - cellstr variables to add to structure (who adds current variables except structure and ans) or variables from structure to add to workspace

Outputs:
-structure - structure array with fields updated from entered variables
-variables - variables evaluated from vars assigned into the caller workspace

Example:
function structure = TestFunction(inarg1,inarg2)
structure = struct;
structure = subfunction1(structure,1,3);
structure = subfunction2(structure,4,6,2);

function structure = subfunction1(structure,x,y)
funpass(structure);
f = x + y;

```
structure=funpass(structure,who);

function structure = subfunction2(structure,a,b,c)
funpass(structure);
z = f*a*b; r = b/c; clear a b c;
structure=funpass(structure,who);

structure =

  x: 1
  y: 3
  f: 4
  z: 96
  r: 3
```

NOTE: To prevent inadvertent errors, it would be best practice to clear input variables before the second calling of funpass.

Created by Justin Theiss

getargs.m

```
[outparams,inparams] = getargs(func)
This function will retreive the out parameters and in parameters for a
function.

Inputs:
fun - string or function handle of function to be parsed
subfun - (optional) string or function handle of subfunction to be parsed

Outputs:
outparames - out argument parameters
inparams - in argument parameters

Example:
[outparams,inparams] = getfunparams('ttest')
outparams = {'h','p','ci','stats'}
inparams = {'x','m','varargin'}

NOTE: This function will only work for functions that have a file listed
when calling functions(func) or built-in matlab functions

requires: subidx

Created by Justin Theiss
```

make_gui.m

```
  data = make_gui(structure)
  This function will create a gui based on a "structure" of gui properties
  structure should be a cell array of structures corresponding to number of
  "pages" for the gui

  Inputs:
  structure - each structure cell should contain the following fields:
  - "name" - the name/title of the corresponding "page"
  - "position" - position of figure (pixel units)
  - uicontrol fields - uicontrol properties (e.g., edit, pushbutton) to use with subfields
  corresponding to properties (e.g., tag, string, position, callback; see example)
  opt (optional) - struct option to be used
  - data - structure to set to guidata for use with callbacks etc.
  - nowait - does not wait for figure to be closed (also prevents setting guidata)
  - nodone - prevents "done" button (still able to be closed)

  Outputs:
  d - structure of fields set from guidata (see example)

  Example:

  INPUT:
  structure{1}.name = 'Info';
  structure{1}.edit.tag = 'age_edit';
  structure{1}.edit.string = 'Enter Age';
  % structure{1}.edit.position = [100,250,100,25];
  structure{1}.pushbutton.string = 'Calculate Birth Year';
  % structure{1}.pushbutton.position = [100,225,200,25];
  structure{1}.pushbutton.callback = {@(x,y)disp(2015-
str2double(get(findobj('tag','age_edit'),'string')))};
  structure{2}.edit.string = 'Enter Name';
  structure{2}.edit(2).tag = 'food_edit';
  structure{2}.edit(2).string = 'Favorite Food';

  FUNCTION:
  d = make_gui(structure);

  OUTPUT:
  d.age_edit = '24';
  d.gui_2_edit_1 = 'Justin';
  d.food_edit = 'smores';

  Note: if no 'callback' is listed, the default callback creates a variable
  (name = tag or 'gui_#_type_#') which is stored in the guidata of the gcf.
```

Note2: if no 'position' properties are included for a "page", the objects
will be distributed starting at the top left corner

Note3: if 'nodone' is used, data will be an empty structure (or equal to
the opt.guidata)

Created by Justin Theiss

match_string.m

match string
[gcstr,idx] = match_string(str)
This function will find the greatest common string derivative of str
that matches the greatest number of strings in cellstr

Input:
str - a cell array of strings to be matched

Output:
gcstr - cell array of greatest common string derivative found in all cellstr
idx - cell array of index of cellstr where gcstr matched

Example:
str = {'J:\This','J:\That','J:\Where','J:\Why'};
[gcstr,idx] = match_string(str);
gcstr = {'J:\'}; idx = {[1 1 1 1]};

requires: subidx

Created by Justin Theiss

printres.m

[hres,fres,outtxt] = printres(varargin)
Create Results Figure

Inputs:
one argument input
title (optional) - title to name results figure when first creating

two arguments input
text - text as string to print to results figure
hres - handle of text object

three arguments input
savepath - fullfile path to save output .txt file
hres - handle of text object
'save' - indicate to save output .txt file

Outputs:
hres - handle for the text object
fres - handle for figure
outtxt - fullfile of saved output .txt file

example:
hres = printres('title'); % creates results figure named 'title'
printres('New text', hres); % prints 'New text' to figure

note: this function uses findjobj as well as jScrollPane, but will work without

requires: choose_SubjectArray findjobj

Created by Justin Theiss

savesubjfile.m

sa = savesubjfile(fileName, task, sa)
Saves subjects.mat file and copies previous to backup folder

Inputs:
fileName - filepath to save subject array
task - task name to save in subjects.mat file
sa - subject array to save

Outputs:
sa - subject array

Note: also creates a "Backup" folder in the same folder as the subjects.mat file to save backup subjects.mat files.

Created by Justin Theiss

sawa.m

subject array and wrapper automation (sawa)
This toolbox will allow you to build pipelines for analysis by wrapping any command line, matlab, or batch functions with input variables from subject arrays or otherwise. The main component of this toolbox is the Subject Array (sometimes seen as "sa"), which contains the subject information (e.g., folders, demographic information, etc.) that can be loaded into different functions to automate data analysis.
The toolbox comes with several functions highlighted by an editable batch editor, command line editor, function wrapper.
Furthermore, users may add scripts/functions to the main folder to be used or build function pipelines using by saving presets in the aforementioned

editors.

Inputs (optional):
funcs - cellstr of functions to run (fullpath)
sv - 0/1 indicating whether savedvars will be used
savedvars - cellstr of savedvars to use (fullpath, must match funcs)

Note: if no function is input, the sawa gui will load.

requires: make_gui choose_SubjectArray sawa_setvars sawa_system

Created by Justin Theiss

sawa_cat.m

out = sawa_cat(dim,A1,A2,...)
This function will force the directional concatenation of the set of
inputs A1, A2, etc. by padding inconsistencies with cells.

Inputs:
dim - 1 for vertcat, 2 for horzcat
varargin - the inputs to concatenate

Outputs:
out - the concatenated cell array

Example:
out = sawa_cat(1,{'Cell1','Cell2'},'Test',{'Cell4',5,'Cell6'})
out =
'Cell1' 'Cell2' []
'Test' [] []
'Cell4' 5 'Cell6'

Created by Justin Theiss

sawa_createvars.m

vars = sawa_createvars(varnam,msg,subrun,sa)
Creates variables for specific use in auto_batch, auto_cmd, auto_wrap.

Inputs:
varnam - variable name
msg - optional string message to display in listdlg
subrun - numeric array of subjects to use (optional)
sa - subject array (optional)
Note: if subrun/sa are not entered, user will choose

Outputs:

vars - variable returned

Example:
varnam = 'Resting State Files';
msg = '';
subrun = 1:33;
sa = ocd;
vars = sawa_createvars(varnam,msg,subrun,sa)
[choose "Subject Array"]
[choose "subjFolders 1"]
[enter "/RestingState/Resting*.nii"]
vars = 'sa(i).subjFolders{1}';

requires: choose_fields getargs sawa_subrun

Created by Justin Theiss

sawa_dlmread.m
raw = sawa_dlmread(file,delim)
This function will read csv/txt files and create a cell array based on delimiters.

Input:
-file - file path for .csv, .txt files or actual string to delimit
-delim - (optional) string delimiter (default is |)

Output:
-raw - the raw output in cell array (rows x columns format)

Example:
file.csv =
Test data; Column 2; Column 3;
Data1; Data2; Data3;
file = '/Test/place/file.csv'; delim = ';';
raw = sawa_dlmread(file,delim)
raw =
'Test Data' 'Column 2' 'Column 3'
'Data1' 'Data2' 'Data3'

Created by Justin Theiss

sawa_editor.m
sawa_editor(sawafile, sv, savedvars)
Loads/runs sawafile functions (e.g., auto_batch, auto_cmd, auto_wrap) with make_gui.

Inputs:
cmd - function to be called (i.e. 'load_sawafile','set_environments',
'choose_subjects','save_presets', or 'load/save/run')
varargin - arguments to be sent to call of cmd (in most cases, the
funpass struct of variables to use with fieldnames as variable names)

Outputs:
fp - funpass struct containing various output variables

Note: default cmd is 'load_sawafile', and the sawa file to be loaded should
be a .mat file with the following varaibles (created during call to save_presets):
- structure - the make_gui structure that will be used
- fp - funpass structure with "sawafile", "funcs", and "options"
sawafile is the fullpath to this .mat file, funcs is a cellstr of
functions to run, and options is a cell array of options to use with
funcs (must be equal in size to funcs).
- program - string name of the program to run (e.g., 'auto_batch')
See Batch_Editor.mat, Command_Line.mat, Wrap_Functions.mat for examples.

requires: make_gui funpass printres sawa_subrun

Created by Justin Theiss

sawa_evalchar.m

out = sawa_evalchar(str,expr)
evaluate strings using subject array (or evaluate expr within str)

Inputs:
str - the string containing the expression to be evaluated
expr (optional) - the expression to be evaluated (default is
'sa\([\w\d]+\)\.')
Outputs:
out - the new string with expr replaced by the evaluation

example:
i = 1;
str = 'sa(i).subjFolders{1}\SubFolder\File.nii'
expr = 'sa(i)\..*\}'
out = sawa_evalchar(str,expr);
out = 'J:\Justin\SPM\NIRR001\SubFolder\File.nii'

example 2:
str = 'sa(1).subj,sa(2).age{1}';
out = sawa_evalchar(str);
out = {'Subj001',12};

note: if evaluating two str at once and one at least one is not char,
output will be cell array (see example 2).

Created by Justin Theiss

sawa_evalvars.m

```
valf = sawa_evalvars(val,subrun,sa)
Evaluate variables created from sawa_createvars

Inputs:
val - string, cell array, structure to evaluate/mkdir/spm_select

Outputs:
valf - evaluated val

Example:
sa = struct('subj',{'test1','test2'},'subjFolders',{'/Users/test1','/Users/test2'})
batch.folder = 'sa(i).subjFolders{1}/Analysis';
batch.files = 'sa(i).subjFolders{1}/Run1/*.nii';
batch.dti = 'sa(i).subjFolders{1}/DTI/DTI.nii,inf';
batch.input = 'evalin("caller",output{1,1}{s})';

for i = 1:2
s = i;
output{1,1}{s} = sa(i).subj;
valf = sawa_evalvars(batch)
end

valf =
  folder: '/Users/test1/Analysis % makes dir
   files: {49x1 cell} % returns from /Users/test1/Run1
     dti: {60x1 cell} % gets 4d frames from /Users/test1/DTI/DTI.nii
   input: 'test1'

valf =
  folder: '/Users/test2/Analysis % makes dir
   files: {49x1 cell} % returns from /Users/test2/Run1
     dti: {60x1 cell} % gets 4d frames from /Users/test2/DTI.nii
   input: 'test2'

Note: if val is a string with file separators, valf will be a string with
"" around the files (i.e. for command line purposes).

requires: sawa_evalchar sawa_find sawa_strjoin

Created by Justin Theiss
```

sawa_fileparts.m

outputs = sawa_fileparts(inptus, part, str2rep, repstr)
function to get fileparts of multiple cells, remove parts of all strings

Inputs:
inputs - cell array of strings
part - empty (to use the entire string) or 'path', 'file', or 'ext'
if part is a cell array of strings (e.g., {'path','file'}), will return
multiple outputs (i.e., output{1} = path, output{2} = file)
str2rep - string (or cell of strings) to replace within each input
repstr - string (or cell of strings) that will replace str2rep

Outputs:
varargout - cell array of string results

example:
inputs = {'X:\Task\Data\Subject1\Structural.nii','X:\Task\Data\Subject2\
Structural.nii'};
outputs = sawa_fileparts(inputs, 'path', 'X:\Task\Data\', '');
Would result in:
outputs = {'Subject1', 'Subject2'};

NOTE: to replace a filepart, put a '"' in front and behind
(i.e. str2rep = '"ext"' and repstr = '_1_"ext"';

Created by Justin Theiss

sawa_find.m

[fnd,vals,tags,reps]=sawa_find(fun,search,varargin)
searches array or obj for search using function fun

Inputs:
fun - (optional) any function used as such:
feval(fun,itemstosearch,search). To use ~, fun should be string
(e.g., '~strcmp').
search - can be a string, number or cell array (for multiple arguments).
varargin - inputs for sawa_getfield (i.e., A, irep, itag)

Outputs:
fnd - a numeric array of logicals where the search was true (relating to
indices of sawa_getfield(varargin{:})).
vals - a cell array of the values of true indices
tags - a cell array of the tags of true indices
reps - a cell array of the string representations of true indices

Example1:

```
sa = struct('group',{{'Control'},{''},{''},{''},{'','Control'},{''},{''},{'Control'}});
[fnd,vals,tags,reps] = sawa_find(@strcmp,'Control',sa,'ddt','\.group\{\d+\}$')
fnd =
    1     0     0     0     0     1     0     0     1
vals =
   'Control'    'Control'    'Control'
tags =
   '{1}'    '{2}'    '{1}'
reps =
   'ddt(1).group{1}'    'ddt(5).group{2}'    'ddt(8).group{1}'

Example2:
printres; % creates "Results" figure with two handles containing "string" property
[fnd,vals,tags,reps] = sawa_find(@strfind,'Results',findobj('-
property','string'),'','\.String$')
fnd =
     0    1
vals =
   'Results:'
tags =
   'String'
reps =
   '(2).String'

NOTE: If no varargin is entered, the default is findobj. Additioanlly, if
[] is input as the third varargin (itag), sawa_find will use sawa_getfield to
recursively search through each value that does not return true. In some cases,
this may not return all values if the recursion limit is met. Similarly, when searching
handles with a vague itag (e.g., '\(1\)$'), it is likely that you will return
looped handle referencing (e.g.,
'.Parent.CurrentFigure.Parent.CurrentFigure.Children(1)').

requires: sawa_getfield

Created by Justin Theiss
```

sawa_getfield.m

```
[values, tags, reps] = sawa_getfield(A, irep, itag);
Gets values, tags, and reps (string representations) of structures or objects

Inputs:
A - array, object, or cell array (does not work for numeric handles)
irep - input string representation of array A
itag - input regular expression of a tag or component of array A that you
want to return (default is '.$'). if itag = '', all values will be returned.
refs - (optional) number of times a field tag can be referenced before
```

stopping (see Note2). default is 1

Outputs:
values - the returned values from each getfield(A,itag)
tags - the end tag of the array for value
reps - string representation of getfield(A,itag)

example:
```
sa = struct('ln_k',{{-1.3863},{-2.7474,-2.6552}});
[values,tags,reps] = sawa_getfield(sa,'sa','ln_k\{\d+\}$')
vals =
   [-1.3863]    [-2.7474]    [-2.6552]
tags =
   '{1}'    '{1}'    '{2}'
reps =
   'sa(1).ln_k{1}'    'sa(2).ln_k{1}'    'sa(2).ln_k{2}'
```

NOTE: Searching for itag uses regexp (i.e. 'subj.s' will find 'subj_s',
and 'subj.*s' will find 'subjFolders' or 'subj_s'). Additionally, itag
should use regexp notation (use regexptranslate to automatically input escape characters)
NOTE2: in order to avoid self-referenceing loops in handles, refs is used
as the number of times a field tag (e.g. .UserData) may exist in a single
rep (e.g., .CurrentAxes.UserData.Axes(2).UserData has two refs).
NOTE3: For handles: unless Parent is included in itag, the .Parent field of handles
is not used to avoid infinite loop of ".Parent.Children.Parent".

requires: sawa_cat

Created by Justin Theiss

sawa_savebatchjob.m

savepath = sawa_savebatchjob(savedir, jobname, matlabbatch)
Save Batch Job

Inputs:
savedir - directory to save batch file
jobname - name of job (as savedir/jobs/jobname) to save batch file
matlabbatch - matlabbatch file to save

Outputs:
savepath - fullpath of saved matlabbatch file

Created by Justin Theiss

sawa_screencapture.m

filename = sawa_screencapture(hfig,filename,ext)
This function simply screencaptures a figure using the hgexport function.

Inputs:
hfig - (optional) figure handle to screen capture default is gcf
filename - (optional) filepath and name to save. default is
get(hfig,'Name')
ext - (optional) extension type to save as (e.g. 'png') default is ext of
filename

Outputs:
filename - fullpath filename if successful, otherwise []

Example:
uicontrol(figure,'style','text','string','this is a test','position',[100,100,100,100])
filename = sawa_screencapture(gcf,'output','png')
filename =

/Applications/sawa/output.png

Created by Justin Theiss

sawa_searchdir.m
[files,fullfiles] = sawa_searchdir(fld, search)
search for files or folders within fld

Inputs:
fld - (optional) starting folder to search within. default is pwd
search - (optional) search term to search (regular expression).
default is [], which will return all files

Outputs:
files - files matching search
fullfiles - full path and filenames of matching files

Example:
fld = 'X:\MainFolder'; search = 'spm.*\.img';
[files,fullfiles] = sawa_searchdir(fld,search)
files = 'spmT_0001.img' 'spmT_0001.img' 'spmF_0001.img'
fullfiles = 'X:\MainFolder\spmT_0001.img'
'X:\MainFolder\Subfolder1\SPM\spmT_0001.img'
'X:\MainFolder\Subfolder2\Subsubfolder\SPM2\spmF_0001.img'

Created by Justin Theiss

sawa_searchfile.m

[files, pos] = sawa_searchfile(str,folder,filetype)
search for str within each .m file (default) in folder

Inputs:
str - string (regular expression)
folder - location of scripts to search
filetype - a regular expression to search files (e.g., \.m$)

Output:
files - full file script locations in which str was found
pos - cell array of character position within each func

requires: sawa_searchdir

Created by Justin Theiss

sawa_setbatch.m

[matlabbatch,chngidx,sts]=sawa_setbatch(matlabbatch,val,itemidx,rep,m)
Set matlabbatch structure items to vals.

Inputs:
matlabbatch - batch job
val - values to set to itemidx in matlabbatch
itemidx - item index corresponding to the list in module.contents{1}
rep - index of the cfg_repeat item to repeat for itemidx (if applicable)
m - index of module to set

Outputs:
matlabbatch - current batch structure for matlabbatch cfg system
chngidx - number indicating change from itemidx (i.e. from replicating module components)
sts - numeric array of 1/0 for status of whether each component was set

Created by Justin Theiss

sawa_setcoords.m

k = sawa_setspmcoords(hfig,coords)
This function will change the coordinates for the spm based axes in figure hfig using coordinates coords.

Inputs:
hfig - (optional) figure with spm axes. default is gcf
coords - (optional) coordinates to change to. default is global max

Outputs:

k - 0/1 for failure/success
Example:

sawa_setdeps.m
matlabbatch = sawa_setdeps(prebatch, matlabbatch)
this function will set dependencies for fields that change (e.g. if a dependency is set for Session 1 and new Sessions are added, new dependencies will be added to mirror the Sessions added). If however, the number available dependencies prior function-filling is greater than the number of dependencies set by user, then the user-set dependencies will be used instead.

Inputs:
prebatch - matlabbatch that is set up by user prior to function-filling
matlabbatch - matlabbatch after function-filling

Outputs:
matlabbatch - matlabbatch with function-filled data and appropriate deps

Note: This function assumes that cell components (e.g. Sessions) should be replicated and non-cell components (e.g. Images to Write) should not be replicated.

requires: sawa_find subidx

Created by Justin Theiss

sawa_setfield.m
structure = sawa_setfield(structure,idx,field,sub,varargin)
Set fields for structure indices

Inputs:
structure- input structure
idx - indices of structure to set (or create)
field- field to set for structure array
sub (optional) string representation of subfields/subcells to set (e.g., '.anotherfield' or '{2}').
varargin- value(s) to add to structure.field(sub). if one varargin, all idx set to varargin{1}, else each idx set to respective varargin.

Outputs:
structure - output structure with set fields

example: set second "age" cell for subjects 2:4 to 12
sa(2:4) = sawa_setfield(sa,2:4,'age','{2}',12);
example 2: set mask.Grey.dimension, mask.White.dimension, mask.CSF.dimension,

to 32, 5, 5 respectively
field = 'mask'; sub = strcat('.Setup.',{'Grey','White','CSF'},'.dimension'); val = {32,5,5};
batch = sawa_setfield(batch,1,field,sub,val{:});

Note: neither the structure string rep nor any cells/periods should be included in field (i.e., field = 'field', NOT field = 'structure.field.subfield{1}').

Created by Justin Theiss

sawa_setupjob.m

[matlabbatch, itemidx, str] = sawa_setupjob(matlabbatch, itemidx)
Opens matlabbatch job using batch editor (cfg_ui) and records items to be used as sawa variables as well as the input user data.

Inputs:
matlabbatch - (optional) job to be loaded. default is empty job
itemidx - (optional) item indices to be set as sawa variables. default is empty

Outputs:
matlabbatch - final job returned with user data
itemidx - item indices to be set as sawa variables
str - string of cfg_ui display for chosen modules

Example:
[matlabbatch, itemidx, str] = sawa_setupjob;
[choose "Call MATLAB function" from BasicIO tab]
[choose "New: String" from Inputs]
[press right arrow to set item as sawa variable]
[enter @disp into "Function to be called"]
[close the Batch Editor]

```
matlabbatch{1} =
  cfg_basicio: [1x1 struct]
itemidx =
   [3]
str{1} =
  'Help on: Call MATLAB function                    ...'
  'Inputs                                    ...'
  '----sawa variable----'
  'Outputs                                    ...'
  'Function to be called                          ...'
```

Note: each item index relates to its index in the display (i.e., itemidx 3 relates to str{1}{3}).

Created by Justin Theiss

sawa_setvars.m

savedvars = setvars(mfil)
Set variables for running multiple functions in GUI for either
scripts or .mat functions

Inputs:
mfil - filename of function to set variables for

Outputs:
savedvars - fullfile path for savedvars.mat file containing variables

Example:
function something = test_function(vars)
%NoSetSubrun

%PQ
f = cell2mat(inputdlg('enter the value for f:'));
%PQ
end

Note: put %NoSetVars in function to keep from setting variables,
put %NoSetSubrun to not choose subjects to run. Also, for
scripts/fucntions, any variables placed between two "%PQ"s will be saved
(see example).

requires: funpass sawa_editor sawa_subrun

Created by Justin Theiss

sawa_strjoin.m

str = sawa_strjoin(C, delim)
This function will concatenate any input as string with delimiter.

Inputs:
C - input to concatenate
delim - delimiter to separate string (default is ', ')

Outputs:
str - string

Example:
str = sawa_strjoin({'test',1,struct('test',{1})}, '\n')
str =

test
1
[1x1 struct]

Note: this function uses any2str to convert non-cell/char/double inputs.
Also, see sprintf for list of escape characters (e.g., \\ for \).

requires: any2str

Created by Justin Theiss

sawa_subrun.m

[subrun,sa,task,fileName] = sawa_subrun(sa,subrun,isubrun)
Choose fileName, task, subjects, and refine subjects based on subject fields

Inputs:
- sa (optional): subject array to use (if empty, choose subject array)
- subrun (optional): indices of subjects in subject array to choose from (if empty, choose subjects to run)
- isubrun (optional): indices of subjects in subject array that are already chosen

Outputs:
- subrun: numeric array of subject indices
- sa: subject array to use
- task: task name (string)
- fileName: filepath of subjects.mat file containing subject array

Example 1:
subrun = sawa_subrun(mid)
Choose a field: [age], Choose cell of age: [1]
Enter function for age{1}: [eq]
Enter search for age{1}: [13]
Refine or Add: [Refine]
Choose a field: [group], Choose cell of group: [cancel]
Enter function for group: [ismember]
Enter search for group: [ADHD]
Refine or Add: [Add]
Choose a field: [cancel]
Output is subrun array with indices of subjects in array mid, the first of which are age 13 followed by subjects in the ADHD group.

requires: choose_SubjectArray choose_fields sawa_find

Created by Justin Theiss

sawa_system.m

[sts,msg] = sawa_system(fun,opts)
This function will run "system" but uses the wine function if running a
.exe on mac.

Input:
fun - the function to be run
opts - the options to be included (as a string)

Output:
sts - status (0/1) of the function run
msg - the command output from the function

requires: update_path

Created by Justin Theiss

sawa_xjview.m

sawa_xjview(varargin)
this function will allow for saving images and cluster details from
multiple files using xjview

variables to input can be any of the following:
files - cellstr of full filepaths of images to use
outfld - cell/str/cellstr of folders to save resulting images and cluster details
into. outfld can be empty cell(s) for same as file folders, str/cellstr subfolder,
or str/cellstr fullpath (default is same as file folders)
pval - pvalue to use (default is 0.001, must be less than or equal to 1)
kval - cluster threshold (default is 5, must be greater than 1)
defxyz - x,y,z coordinates to snap to (default is global max)
mask - string of default mask to use (default is 'single T1')
slc - structure with any of following fields for slice view images:
- sep - separation (in mm) of slices
- col - number of columns to display in image
- row - number of rows to display in image
itype - image type for saving images (default is .png)
options:
'othermask' - option to choose other mask
'notxt' - option to not save txt files
'noxls' - option to not save xlsx files
'noimg' - option to not save xjview main images
'noslc' - option to not save slice view images

Example:
sawa_xjvie(fullfile(cd,'spmT_0001.nii'),0.01,124,[12,-8,14],'notxt','noxls')

This example would save xjview main image and slice view images for 'spmT_0001.nii', use a pval of 0.01, kval of 124 voxels, jump to coordinates [12, -8, 14], and save the resulting images into the directory cd.

NOTE: Order of input variables generally does not matter, except that pval, kval, maskchc, and sep must be input in order (e.g., if pval does not exist and sep is entered as .5, pval would errantly be set to .5).

requires: sawa_cat, sawa_screencapture, subidx, xjview

Created by Justin Theiss

sawa_xlsread.m

raw = sawa_xlsread(xfil)
This function is mainly used since xlsread does not work consistently with .xlsx files on mac. However, this will not slow down the ability the functionality on pc or with .xls files. It also simplifies the usual [~,~,raw] = xlsread(xfil,s).

Inputs:
xfil - filename of excel to read all data from
s - (optional) sheet to pull all raw data from (default is 1)

Outputs:
raw - the raw cell data from excel page s

Created by Justin Theiss

settimeleft.m

hobj = settimeleft(varargin)
sets time left display

Inputs:
i - (optional) current iteration
subrun - (optional) numeric array of all iterations
hobj - (optional) handle for settimeleft obj
wmsg - (optional) message to update

Outputs:
hobj - handle for settimeleft obj

Example:
h = settimeleft;
for x = doforloop
\\stuff\\

settimeleft(x, doforloop, h, 'optional text');
end

Note: the tag for the hobj is set to SAWA_WAITBAR. Also, the waitbar automatically closes once the final iteration has completed (i.e. i = subrun(end)).

requires: subidx

Created by Justin Theiss

subidx.m

out = subidx(item,idx)
Index item as item(idx)

Inputs:
item - object to be indexed
idx - string or numeric index to use
bnd - (optional) string representing the type of boundaries to be used when evaluating (e.g., '[]' or '{}'). Default is '[]'.

Outputs:
out - returned object that is item(idx)

example1:
out=subidx(class('test'),1:3)
out='cha'
example2:
out=subidx(regexp(report,'Cluster\s(?<names>\d+)','names'),'.names','{}')
out={'1','2'}
out=subidx(regexp(report,'Cluster\s(?<names>\d+)','names'),'.names')
out='12'
example3:
curdir = '/Volumes/J_Drive/TestFolder/TestFile.mat'
out=subidx('fileparts(curdir)','varargout{2}')
out='TestFile'

Note: to index and output argument of a function, enter item as a string to be evaluated and idx as 'varargout{index}' (see example 3).
Note2: bnd is only really applicable for indexing structure fields or other non-traditional indexing (see example 2).

Created by Justin Theiss

subjectarray.m

```
[sa, subrun] = subjectarray(cmd, varargin)
This is the first step to using the Subject Array and Study Organizer.

Inputs:
cmd - cellstr array of subfunction(s) to run
varargin - arguments to pass to subfunction
if no inputs, subjectarray will run its gui (see example)

Outputs (only returned from command prompt call):
sa - subject array
subrun - subject indices of sa

Example1 (no inputs):
- Subject Array Name: Enter a name for the subject array (e.g., gonogo).
- Subjects.mat File: This is the file that will hold the subject array.
You may add a subject array to a previous subjects.mat file or create one.
- Enter Subjects: Choose subjects/add subjects to subject array.
- Subject Folders: Choose the main folders for each subject (and
optionally create them)
- Create New Field: Create a field for each subject in the subject array
(e.g., age, group, gender, etc.).
- Load/Save Subject Array:
-- Load Subject Array: Load a previous subjects.mat file or an excel, txt
or mat file. (Note: excel files should have field names as headers and
one header must be 'subj' with subject names. txt files should be the
same with tabs between columns. mat files should contain field names as
variables with rows corresponding to each subject.) Once loaded, subject
arrays may be edited.
-- Save Subject Array: Save the subject array to the chosen subjects.mat
file. Additionally, you may save the subject array as an excel, txt, or
mat file in the format as indicated above.

Example2 (with inputs):
[sa,subrun] = subjectarray('load_sa',struct(fileName,{'/Users/test.mat'},'flds',...
{{'subj','subjFolders1'}},'subrun',{[2,3]}));
sa =
1x4 struct array with fields:

  subj
  subjFolders

subrun =

  2   3
```

requires: make_gui cell2strtable choose_SubjectArray choose_fields funpass printres savesubjfile sawa sawa_createvars sawa_dlmread sawa_getfield sawa_setfield sawa_strjoin sawa_subrun sawa_xlsread update_array

Created by Justin Theiss

update_array.m

sa = update_array(task)

Used to update array (sa) with the latest data. Primarily used when savedvars is chosen, to ensure that the array is not going to save older data.

Inputs:
task - string of task to update

Outputs:
sa - subject array updated

Example:
task = 'ddt';
load(savedvars); sa = update_array(task);

Created by Justin Theiss

update_path.m

update_path(fil,mfil)
This function will update a filepath (fil) for the .m file entered (i.e. mfilename('fullpath'))

Inputs:
fil - file path or folder path as a variable (see example)
mfil - mfilename('fullpath') for the .m file to edit
filvar (optional) - if fil is a cellstr (e.g., fil{1} = '';) then filvar should be the str representation of fil (i.e. filvar = 'fil';)

Outputs:
new_fil - the updated file/folder path

Example:
test_dir = 'C:\Program Files\Deleted Folder\SomeProgram';
mfil = 'C:\Program Files\sawa\Test\SomeScript';
update_path(test_dir,mfil);
The script "SomeScript" will now have been rewritten with the updated path for test_dir

NOTE: within the script file (mfil), fil must be defined as follows:
fil = 'filepath';
If the single quotes and semicolon are missing, update_path will not work.
Furthermore, if there are multiple "fil = 'filepath';", only the first
will be used.

Created by Justin Theiss