Implement, export and import groups of subjects

The BRAPH 2 Developers

September 20, 2023

This is the developer tutorial for implementing, exporting and importing groups of subjects with a certain type of data.

In this Tutorial, we will explain how to create generator files `*.gen.m` for new subjects and well as exporting and importing subjects files. All `*.gen.m` files can then be compiled by `braph2genesis`.

All types of subjects are extensions of the base element `Subject`. In this tutorial, we will use as examples the subjects `SubjectCON` (subject with connectivity data), `SubjectCON_MP` (subject with connectivity multiplex data), `SubjectFUN` (subject with functional data), `SubjectFUN_MP` (subject with functional multiplex data), `SubjectST` (subject with structural data), and `SubjectST_MP` (subject with structural multiplex data).

Furthermore, all types of exporters and importers are extensions of the base elements `Exporter` and `Importer` respectively. Here, we will use as examples `ExporterGroupSubjectCON_TXT` (exports a group of subjects with connectivity data to a series of TXT file), `ExporterGroupSubjectCON_XLS` (exports a group of subjects with connectivity data to a series of XLSX file), `ImporterGroupSubjectCON_TXT` (imports a group of subjects with connectivity data to a series of TXT file) and `ImporterGroupSubjectCON_XLS` (imports a group of subjects with connectivity data to a series of XLSX file).

Contents

Implementation of a subject with connectivity data

Subject with connectivity data (SubjectCON)

We will start by implementing in detail `SubjectCON`. The connectivity matrix can be obtained from DTI data.

Code 1: **SubjectCON element header.** The `header` section of the generator code for `_SubjectCON.gen.m` provides the general information about the `SubjectCON` element.

```
%% iheader!
SubjectCON < Subject (sub, subject with connectivity matrix) is a subject
     with connectivity matrix (e.g. DTI). (1)

%%% idescription!
Subject with a connectivity matrix (e.g. obtained from DTI).

%%% iseealso! (2)
ImporterGroupSubjectFUN_TXT, ExporterGroupSubjectFUN_TXT,
     ImporterGroupSubjectFUN_XLS, ExporterGroupSubjectFUN_XLS
```

(1) The element `SubjectCON` is defined as a subclass of `Subject`. The moniker will be `sub`.

(2) Other related functions.

Code 2: **SubjectCON element prop update.** The `props_update` section of the generator code for `_SubjectCON.gen.m` updates the properties of the `Subject` element. This defines the core properties of the subject.

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the subject.
%%%% idefault!
'SubjectCON'

%%% iprop!
DESCRIPTION (constant, string) is the description of the subject.
%%%% idefault!
'SubjectCON with a connectivity matrix (e.g. obtained from DTI).'

%%% iprop!
TEMPLATE (parameter, item) is the template of the subject.
%%% isettings!
'SubjectCON'

%%% iprop!
ID (data, string) is a few-letter code for the subject.
%%%% idefault!
'SubjectCON ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the subject.
%%%% idefault!
'SubjectCON label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the subject.
%%%% idefault!
```

```
'SubjectCON notes'

%% iprops!

%%% iprop!
BA (data, item) is a brain atlas.
%%%% isettings!
'BrainAtlas'


%%% iprop!
CON (data, smatrix) is an adjacency matrix.
%%%% icheck_value!
br_number = sub.get('BA').get('BR_DICT').get('LENGTH'); (1)
check = isequal(size(value), [br_number, br_number]);  (2)
if check  (3)
    msg = 'All ok!';
else
    msg = ['CON must be a square matrix with the dimension equal to the
     number of brain regions (' int2str(br_number) ').'];
end

%%%% igui! (4)
pr = PanelPropMatrix('EL', sub, 'PROP', SubjectCON.CON, ...
    'ROWNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
    'COLUMNNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
    varargin{:});
```

(1) defines the number of brain regions from the Brain Atlas.
(2) The `value` is the data matrix. Checking the size of `value` is equal to the number of brain regions.
(3) returns the check information `msg` according to the variable `check`.
(4) plots the panel of a property matrix-like with element `sub` and the property number `SubjectCon.Con`. `ROWNAME` and `COLUMNNAME` are the name of regions from brain atlas.

Code 3: **SubjectCON element tests.** The `tests` section from the element generator `_SubjectCON.gen.m`. A general test should be prepared to test the properties of the Subject when it is empty and full. Furthermore, additional tests should be prepared for the rules defined.

```
%% itests!

%%% itest!
%%%% iname!
GUI (1)
%%%% iprobability! (2)
.01
%%%% icode!
im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx'); (3)
ba = im_ba.get('BA'); (4)

gr = Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('
     IT_CLASS', 'SubjectCON')); (5)
for i = 1:1:50 (6)
    sub = SubjectCON( ... (7)
        'ID', ['SUB CON ' int2str(i)], ...
        'LABEL', ['Subejct CON ' int2str(i)], ...
        'NOTES', ['Notes on subject CON ' int2str(i)], ...
        'BA', ba, ...
        'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
        );
    sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
     rand())) (8)
    sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
     CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1))) (9)
    gr.get('SUB_DICT').get('ADD', sub) (10)
end

gui = GUIElement('PE', gr, 'CLOSEREQ', false); (11)
gui.get('DRAW') (12)
gui.get('SHOW') (13)

gui.get('CLOSE') (14)
```

(1) checks that GUI is constructing well.

(2) assigns a low test execution probability.

(3) imports the brain atlas `desikan` from the file `'desikan_atlas.xlsx'`.There are also other atlases in Braph2 folder `atlases`, including `aal90_atlas.xlsx`, `aal116_atlas.xlsx`, `bna_atlas.xlsx`, `craddock_atlas.xlsx`, `desikan_subcortical_atlas.xlsx`, `destrieux_atlas.xlsx`, `destrieux_subcortical_atlas.xlsx`, `schaefer200_atlas.xlsx` and `subcortical_atlas.xlsx`.

(4) returns the brain atlas.

(5) represents a group of subjects whose class is defined in the property `'SUB_CLASS'`. `'SUB_DICT'` manages the subjects as an indexed dictionary of subjects.

(6) construts 50 subjects.

(7) defines the `'ID'`, `'LABEL'`, `'NOTES'`, `'BA'` (Brain Atlas) and `'CON'` (a random adjacency matrix) for a subject.

(8) adds a random `Numeric` `'Age'` as the variable of interest of the subject.

(9) adds a random `Categoric` `'Sex'` as the variable of interest of the subject.

(10) adds `'sub'` into group.

(11) constructs the GUI panel from `gr`. Setting the `'CLOSEREQ'` to `false` means doesn't confirm whether the GUI is close.

(12) draws the contents of a GUI before showing it.

(13) shows the figure and its dependent figures.

(14) closes the figure and its dependent figures.

Subject with connectivity multiplex data (SubjectCON_MP)

We can now use `SubjectCON` as the basis to implement the `SubjectCON_MP`. The parts of the code that are modified are highlighted. The multilayer data allows connections between any nodes across the multiple layers. The `SubjectCON_MP` can also be used on ordinal multilayer data.

Code 4: **SubjectCON_MP element header.** The `header` section of the generator code for `_SubjectCON_MP.gen.m` provides the general information about the `SubjectCON_MP` element. ← Code 1

```
%% iheader!
SubjectCON_MP < Subject (sub, subject with connectivity multiplex data) is a
      subject with connectivity multiplex data.

%%% idescription!
Subject with L connectivity matrices (e.g. obtained from DTI).

%%% iseealso!
ImporterGroupSubjectCON_MP_TXT, ExporterGroupSubjectCON_MP_TXT,
     ImporterGroupSubjectCON_MP_XLS, ExporterGroupSubjectCON_MP_XLS
```

Code 5: **SubjectCON_MP element prop update.** The `props_update` section of the generator code for `_SubjectCON_MP.gen.m` updates the properties of the `Subject` element. ← Code 2

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the subject.
%%%% idefault!
'SubjectCON_MP'

%%% iprop!
DESCRIPTION (constant, string) is the description of the subject.
%%%% idefault!
'Subject with L connectivity matrices (e.g. obtained from DTI).'

%%% iprop!
TEMPLATE (parameter, item) is the template of the subject.
%%% isettings!
'SubjectCON_MP'

%%% iprop!
ID (data, string) is a few-letter code for the subject.
%%%% idefault!
'SubjectCON_MP ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the subject.
%%%% idefault!
'SubjectCON_MP label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the subject.
%%%% idefault!
```

```
'SubjectCON_MP notes'

%% iprops!

%%% iprop!
BA (data, item) is a brain atlas.
%%%% isettings!
'BrainAtlas'

%%% iprop!
L (data, scalar) is the number of layers of subject data. (1)
%%%% idefault!
2 (2)

%%% iprop!
LAYERLABELS (metadata, stringlist) are the layer labels provided by the user . (3)

%%% iprop!
ALAYERLABELS (query, stringlist) returns the processed layer labels. (4)
%%%% icalculate!
value = sub.get('LAYERLABELS'); (5)

%%% iprop!
CON_MP (data, cell) is a cell containing L matrices corresponding connectivity matrices of each layer.
%%%% icheck_value!
br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
num_layers = sub.get('L'); (6)
check = (iscell(value) && isequal(length(value), num_layers)  && isequal( cellfun(@(v) size(v, 1), value), ones(1, num_layers) * br_number)  && isequal( cellfun(@(v) size(v, 2), value), ones(1, num_layers) * br_number)) || (isempty(value) && br_number == 0); (7)
if check
    msg = 'All ok!';
else
    msg = ['CON_MP must be a cell with L square matrices with the dimension equal to the number of brain regions (' int2str(br_number) ').'];
end
%%%% igui!
pr = PanelPropCell('EL', sub, 'PROP', SubjectCON_MP.CON_MP, ...
    'TABLE_HEIGHT', s(40), ... (8)
    'XSLIDERSHOW', true, ... (9)
    'XSLIDERLABELS', sub.getCallback('ALAYERLABELS'), ... (10)
    'YSLIDERSHOW', false, ... (11)
    'ROWNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
    'COLUMNNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
    varargin{:});
```

(1) defines a parameter to determine the number of layers of subject data. This property must be of a scalar parameter.

(2) defines the default option, in this case '2'.

(3) defines a parameter to determine the labels for each layer. This property must be of string list parameter.

(4) defines a parameter to determine the processed labels for each layer. This property must be of string list parameter.

(5) defines the value from the property 'LAYERLABELS' of SubjectCON_MP.

(6) defines the number of layers.

(7) checks the size of each layer is equal to the number of brain regions.

(8) defines the height of table.

(9) defines the option of showing in X-axis slider.

(10) defines the X-axis sliders' labels.

(11) defines the option of not showing in Y-axis slider.

Code 6: **SubjectCON_MP element tests.** The `tests` section from the element generator `_SubjectCON_MP.gen.m`. ← Code 3

```
%% itests!

%%% itest!
%%%% iname!
GUI
```

```
%%%% !probability!
.01
%%%% !code!
im_ba = ImporterBrainAtlasXLS('FILE', 'aal90_atlas.xlsx');
ba = im_ba.get('BA');

gr = Group('SUB_CLASS', 'SubjectCON_MP', 'SUB_DICT', IndexedDictionary('
     IT_CLASS', 'SubjectCON_MP'));
for i = 1:1:10
    sub = SubjectCON_MP( ...
        'ID', ['SUB CON_MP ' int2str(i)], ...
        'LABEL', ['Subejct CON_MP ' int2str(i)], ...
        'NOTES', ['Notes on subject CON_MP ' int2str(i)], ...
        'BA', ba, ...
        'L', 3, ... (1)
        'LAYERLABELS', {'L1' 'L2' 'L3'}, ... (2)
        'CON_MP', {rand(ba.get('BR_DICT').get('LENGTH')), rand(ba.get('
     BR_DICT').get('LENGTH')), rand(ba.get('BR_DICT').get('LENGTH'))} ...
        ); (3)
    sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
     rand()))
    sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
     CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
    gr.get('SUB_DICT').get('ADD', sub)
end

gui = GUIElement('PE', gr, 'CLOSEREQ', false);
gui.get('DRAW')
gui.get('SHOW')

gui.get('CLOSE')
```

(1) defines the number of layers.

(2) defines the label of each layer.

(3) constructs 3 layers randomly with size of brain regions by brain regions.

Implementation of Importer and Exporter of the data (for Subject-CON)

Importer of a CON subject group from TXT (ImporterGroupSubjectCON_TXT)

We will start by implementing in detail `ImporterGroupSubjectCON_TXT`. The data should be stored in the folder 'Group1' and 'Group2', and the file format is '.txt'.

Code 7: **ImporterGroupSubjectCON_TXT element header.** The `header` section of the generator code for `_ImporterGroupSubjectCON_TXT.gen.m` provides the general information about the `Importer` element.

```
1
2 %% !header!
3 ImporterGroupSubjectCON_TXT < Importer (im, importer of CON subject group
      from TXT) imports a group of subjects with connectivity data from a
      series of TXT files. (1)
4
5 %%% !description!
6 ImporterGroupSubjectCON_XLS imports a group of subjects with connectivity
      data from a series of XLS/XLSX files contained in a folder named "
      GROUP_ID". All these files must be in the same folder; also, no other
      files should be in the folder. Each file contains a table of values
      corresponding to the adjacency matrix. The variables of interest are
      from another XLS/XLSX file named "GROUP_ID.vois.xlsx" (if exisitng)
      consisting of the following columns: Subject ID (column 1), covariates
      (subsequent columns). The 1st row contains the headers, the 2nd row a
      string with the categorical variables of interest, and each subsequent
      row the values for each subject.
7
8 %%% !seealso!
9 Group, SunbjectCON, ExporterGroupSubjectCON_TXT
```

(1) The element `ImporterGroupSubjectCON_TXT` is defined as a subclass of `Importer`. The moniker will be `im`.

Code 8: **ImporterGroupSubjectCON_TXT element prop update.** The `props_update` section of the generator code for `_ImporterGroupSubjectCON_TXT.gen.m` updates the properties of the `Importer` element.

```
1
2 %% !props_update!
3
4 %%% !prop!
5 NAME (constant, string) is the name of the CON subject group importer from
      TXT.
6 %%%% !default!
7 'ImporterGroupSubjectCON_TXT'
8
9 %%% !prop!
10 DESCRIPTION (constant, string) is the description of the CON subject group
      importer from TXT.
11 %%%% !default!
12 'ImporterGroupSubjectCON_TXT imports a group of subjects with connectivity
      data from a series of TXT file and their covariates (optional) from
      another TXT file.'
```

```
13
14 %%% iprop!
15 TEMPLATE (parameter, item) is the template of the CON subject group importer
       from TXT.
16 %%%% isettings!
17 'ImporterGroupSubjectCON_TXT'
18
19 %%% iprop!
20 ID (data, string) is a few-letter code for the CON subject group importer
      from TXT.
21 %%%% idefault!
22 'ImporterGroupSubjectCON_TXT ID'
23
24 %%% iprop!
25 LABEL (metadata, string) is an extended label of the CON subject group
      importer from TXT.
26 %%%% idefault!
27 'ImporterGroupSubjectCON_TXT label'
28
29 %%% iprop!
30 NOTES (metadata, string) are some specific notes about the CON subject group
       importer from TXT.
31 %%%% idefault!
32 'ImporterGroupSubjectCON_TXT notes'
33
34 %% iprops!
35
36 %%% iprop!
37 DIRECTORY (data, string) is the directory containing the CON subject group
      files from which to load the subject group.
38 %%%% idefault!
39 fileparts(which('test_braph2'))
40
41 %%% iprop!
42 GET_DIR (query, item) opens a dialog box to set the directory from where to
      load the TXT files of the CON subject group.
43 %%%% isettings!
44 'ImporterGroupSubjectCON_TXT'
45 %%%% icalculate!
46 directory = uigetdir('Select directory'); (1)
47 if ischar(directory) && isfolder(directory)
48   im.set('DIRECTORY', directory); (2)
49 end
50 value = im;
51
52 %%% iprop!
53 BA (data, item) is a brain atlas.
54 %%%% isettings!
55 'BrainAtlas'
56
57 %%% iprop!
58 GR (result, item) is a group of subjects with connectivity data.
59 %%%% isettings!
60 'Group'
61 %%%% icheck_value!
62 check = any(strcmp(value.get(Group.SUB_CLASS_TAG), subclasses('SubjectCON',
      [], [], true))); (3)
63 %%%% idefault!
64 Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', '
      SubjectCON')) (4)
```

(1) selects folder firectory that contains txt data.

(2) saves the folder firectory into the `'DIRECTORY'` property of `im`.

(3) checks that the class of subjects of the group is the same as `'SubjectCON'`.

(4) represents a group of subjects whose class is defined in the property `'SUB_CLASS'`. `'SUB_DICT'` manages the subjects as an indexed dictionary of subjects.

```
65
66  %%%% !calculate! (5)
67  gr = Group( ...
68      'SUB_CLASS', 'SubjectCON', ...
69      'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON') ...
70      );
71
72  gr.lock('SUB_CLASS'); (6)
73
74  directory = im.get('DIRECTORY'); (7)
75  if isfolder(directory) (8)
76      wb = braph2waitbar(im.get('WAITBAR'), 0, 'Reading directory ...'); (9)
77
78      [~, name] = fileparts(directory); (10)
79      gr.set( ... (11)
80          'ID', name, ...
81          'LABEL', name, ...
82          'NOTES', ['Group loaded from ' directory] ...
83          );
84
85      try
86          braph2waitbar(wb, .15, 'Loading subjecy group ...')
87
88          files = dir(fullfile(directory, '*.txt')); (12)
89
90          if ~isempty(files)
91              % brain atlas
92              ba = im.get('BA'); (13)
93              if ba.get('BR_DICT').get('LENGTH') == 0 (14)
94                  br_number = size(readtable(fullfile(directory, files(1).name
     ), 'Delimiter', '\t'), 1); (15)
95                  br_dict = ba.memorize('BR_DICT');
96                  for j = 1:1:br_number
97                      br_dict.get('ADD', BrainRegion('ID', ['br' int2str(j)]))
     (16)
98                  end
99              end
100
101
102             sub_dict = gr.memorize('SUB_DICT'); (17)
103             for i = 1:1:length(files)
104                 braph2waitbar(wb, .15 + .85 * i / length(files), ['Loading
     subject ' num2str(i) ' of ' num2str(length(files)) ' ...']) (18)
105
106                 [~, sub_id] = fileparts(files(i).name);
107                 CON = table2array(readtable(fullfile(directory, files(i).
     name), 'Delimiter', '\t')); (19)
108                 if size(CON, 1) ~= ba.get('BR_DICT').get('LENGTH') || size(
     CON, 2) ~= ba.get('BR_DICT').get('LENGTH') (20)
109                     error( ...
110                         [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO], ...
111                         [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO '\\n'
     ...
112                         'The file ' sub_id ' should contain a matrix '
```

(5) constructs an empty `Group`.

(6) locks the property `'SUB_CLASS'` irreversibly.

(7) returns the data directory from previous saving (2).

(8) checks that folder exists.

(9) creates the waitbar with an initial progress of `0` displaying `'Reading directory ...'`.

(10) returns the folder name from folder directory.

(11) sets the properties `'ID'`, `'LABEL'` and `'NOTES'` for Group.

(12) finds all .txt files in the `directory`.

(13) returns the brain atlas.

(14) checks that the number of nodes in brain atlas is equal to 0.

(15) adds the number of regions of the first file to the brain atlas.

(16) adds the `'ID'` of each brain region.

(17) adds the subjects.

(18) updates the waitbar for each file.

(19) reads each file with a delimiter specified in `Delimiter`.

(20) checks that the number of the nodes in file is equal to the number of nodes in atlas.

```
        int2str(ba.get('BR_DICT').get('LENGTH')) 'x' int2str(ba.get('BR_DICT').
        get('LENGTH')) ', ' ...
113                     'while it is ' int2str(size(CON, 1)) 'x' int2str(
        size(CON, 2)) '.'] ...
114                     ) (21)
115             end
116
117             sub = SubjectCON( ...
118                 'ID', sub_id, ...
119                 'BA', ba, ...
120                 'CON', CON ...
121             );
122             sub_dict.get('ADD', sub);
123         end
124
125
126         if isfile([directory '.vois.txt']) (22)
127             vois = textread([directory '.vois.txt'], '%s', 'delimiter',
        '\t', 'whitespace', ''); (23)
128             vois = reshape(vois, find(strcmp('', vois), 1) - 1, [])';
        (24)
129             for i = 3:1:size(vois, 1)
130                 sub_id = vois{i, 1};
131                 sub = sub_dict.get('IT', sub_id);
132                 for v = 2:1:size(vois, 2)
133                     voi_id = vois{1, v};
134                     if isempty(vois{2, v}) (25)
135                         sub.memorize('VOI_DICT').get('ADD', ...
136                             VOINumeric( ...
137                                 'ID', voi_id, ...
138                                 'V', str2num(vois{i, v}) ...
139                                 ) ...
140                             );(26)
141                     elseif ~isempty(vois{2, v}) (27)
142                         categories = eval(vois{2, v});
143                         sub.memorize('VOI_DICT').get('ADD', ...
144                             VOICategoric( ...
145                                 'ID', voi_id, ...
146                                 'CATEGORIES', str2cell(categories), ...
147                                 'V', find(strcmp(vois{i, v}, categories)
        ) ...
148                                 ) ...
149                             );
150                     end
151                 end
152             end
153         end
154     end
155 catch e
156     braph2waitbar(wb, 'close')
157
158     rethrow(e)
159 end
160
161 braph2waitbar(wb, 'close') (28)
162 else
163     error([BRAPH2.STR ':ImporterGroupSubjectCON_TXT:' BRAPH2.ERR_IO], ...
```

(21) outputs the error information.

(22) adds the variables of interest (`vois`).

(23) reads the file `vois.txt`.

(24) reshape the `vois`.

(25) checks that the variable is Numeric.

(26) adds the variable of interest witt `'ID'` and value `'V'`.

(27) checks that the variable is CATEGORIES.

(28) closes the waitbar.

```
164         [BRAPH2.STR ':ImporterGroupSubjectCON_TXT:' BRAPH2.ERR_IO '\\n' ...
165         'The prop DIRECTORY must be an existing directory, but it is '''
        directory '''.'] ...
166         );
167 end
168
169 value = gr;
```

Code 9: **ImporterGroupSubjectCON_TXT element tests.** The `tests` section from the element generator _ImporterGroupSubjectCON_TXT.gen.m. In this section, example data are created for testing.

```
%% itests!

%%% iexcluded_props! (1)
[ImporterGroupSubjectCON_TXT.GET_DIR]

%%% itest!
%%%% iname!
Create example files (2)
%%%% icode!
data_dir = [fileparts(which('SubjectCON')) filesep 'Example data CON TXT'];
    (3)
if ~isdir(data_dir)
    mkdir(data_dir); (4)

    % Brain Atlas
    im_ba = ImporterBrainAtlasTXT('FILE', 'desikan_atlas.txt'); (5)
    ba = im_ba.get('BA');
    ex_ba = ExporterBrainAtlasTXT( ... (6)
        'BA', ba, ...
        'FILE', [data_dir filesep() 'atlas.txt'] ...
        );
    ex_ba.get('SAVE')
    N = ba.get('BR_DICT').get('LENGTH'); (7)

    % saves RNG
    rng_settings_ = rng(); rng('default') (8)
\code{rng}
    sex_options = {'Female' 'Male'};

    % Group 1 (9)
    K1 = 2; (10)
    beta1 = 0.3; (11)
    gr1_name = 'CON_Group_1_TXT';
    gr1_dir = [data_dir filesep() gr1_name];
    mkdir(gr1_dir);
    vois1 = [ (12)
        {{'Subject ID'} {'Age'} {'Sex'}}
        {{} {} {['{' sprintf(' ''%s'' ', sex_options{:}) '}']}}
        ];
    for i = 1:1:50 % subject number (13)
        sub_id = ['SubjectCON_MP_' num2str(i)];

        h1 = WattsStrogatz(N, K1, beta1); (14)

        A1 = full(adjacency(h1)); A1(1:length(A1)+1:numel(A1)) = 0; (15)
        r = 0 + (0.5 - 0)*rand(size(A1)); diffA = A1 - r; A1(A1 ~= 0) =
     diffA(A1 ~= 0); (16)
        A1 = max(A1, transpose(A1)); (17)
```

(1) List of properties that are excluded from testing.

(2) creates the example files.

(3) assigns the example directory `'Example data CON TXT'`.

(4) makes the example directory.

(5) imports the brain atlas.

(6) exports the brain atlas as file 'atlas.txt'.

(7) returns the number of brain regions.

(8) sets the random number generator (`rng`) to `'default'`.

(9) generates the data for group1.

(10) assigns the degree (mean node degree is 2K) for group 1.

(11) assigns the rewiring probability for group 1.

(12) assigns the header with `'Subject ID'`, `'Age'`, and `'Sex'`.

(13) generates 50 subjects.

(14) creates WS graph.

(15) extracts the adjacency matrix.

(16) makes the adjacency matrix weighted.

(17) makes the adjacency matrix symmetric.

```
48
49          writetable(array2table(A1), [gr1_dir filesep() sub_id '.txt'], '
        Delimiter', '\t', 'WriteVariableNames', false) (18)
50
51          vois1 = [vois1; {sub_id, randi(90), sex_options(randi(2))}]; (19)
52      end
53      writetable(table(vois1), [data_dir filesep() gr1_name '.vois.txt'], '
        Delimiter', '\t', 'WriteVariableNames', false) (20)
54
55      % Group 2 (21)
56      K2 = 2;
57      beta2 = 0.85;
58      gr2_name = 'CON_Group_2_TXT';
59      gr2_dir = [data_dir filesep() gr2_name];
60      mkdir(gr2_dir);
61      vois2 = [
62          {{'Subject ID'} {'Age'} {'Sex'}}
63          {{} {} {['{' sprintf(' ''%s'' ', sex_options{:}) '}']}}
64          ];
65      for i = 51:1:100
66          sub_id = ['SubjectCON_MP_' num2str(i)];
67
68          h2 = WattsStrogatz(N, K2, beta2);
69
70          A2 = full(adjacency(h2)); A2(1:length(A2)+1:numel(A2)) = 0;
71          r = 0 + (0.5 - 0)*rand(size(A2)); diffA = A2 - r; A2(A2 ~= 0) =
        diffA(A2 ~= 0);
72          A2 = max(A2, transpose(A2));
73
74          writetable(array2table(A2), [gr2_dir filesep() 'SubjectCON_' num2str
        (i) '.txt'], 'Delimiter', '\t', 'WriteVariableNames', false)
75
76          % variables of interest
77          vois2 = [vois2; {sub_id, randi(90), sex_options(randi(2))}];
78      end
79      writetable(table(vois2), [data_dir filesep() gr2_name '.vois.txt'], '
        Delimiter', '\t', 'WriteVariableNames', false)
80
81      % reset RNG
82      rng(rng_settings_) (22)
83  end
84
85  %%% itest_functions!
86  function h = WattsStrogatz(N,K,beta) (23)
87  % H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
88  % nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
89  %
90  % beta = 0 is a ring lattice, and beta = 1 is a random graph.
91
92  % Connect each node to its K next and previous neighbors. This constructs
93  % indices for a ring lattice.
94  s = repelem((1:N)',1,K); (24)
95  t = s + repmat(1:K,N,1);  (25)
96  t = mod(t-1,N)+1; (26)
97
98  for source=1:N (27)
```

(18) writes the matrix into the file.

(19) creates the variables of interest.

(20) writes the variables of interest.

(21) generates the data for group 2.

(22) resets random number generator.

(23) defines a function named `WattsStrogatz` that takes three input arguments: `N` (number of nodes), `K` (number of neighbors for each node), and `beta` (rewiring probability).

(24) creates a matrix `s` where each row corresponds to a node, and each column contains the node's number repeated K times.

(25) calculates the target nodes for each node in the ring lattice.

(26) ensures that the indices wrap around, creating a circular lattice.

(27) rewires the target node of each edge with probability `beta`.

```
99      switchEdge = rand(K, 1) < beta; (28)
100
101     newTargets = rand(N, 1); (29)
102     newTargets(source) = 0;
103     newTargets(s(t==source)) = 0;
104     newTargets(t(source, ~switchEdge)) = 0;
105
106     [~, ind] = sort(newTargets, 'descend');
107     t(source, switchEdge) = ind(1:nnz(switchEdge)); (30)
108 end
109
110 h = graph(s,t); (31)
111 end
112
113 %%% itest!
114 %%%% iname!
115 GUI
116 %%%% iprobability!
117 .01
118 %%%% icode!
119 im_ba = ImporterBrainAtlasTXT('FILE', [fileparts(which('SubjectCON'))
        filesep 'Example data CON TXT' filesep 'atlas.txt']);
120 ba = im_ba.get('BA');
121
122 im_gr = ImporterGroupSubjectCON_TXT( ... (32)
123     'DIRECTORY', [fileparts(which('SubjectCON')) filesep 'Example data CON
        TXT' filesep 'CON_Group_1_TXT'], ...
124     'BA', ba, ...
125     'WAITBAR', true ...
126     );
127 gr = im_gr.get('GR'); (33)
128
129 gui = GUIElement('PE', gr, 'CLOSEREQ', false); (34)
130 gui.get('DRAW')
131 gui.get('SHOW')
132
133 gui.get('CLOSE')
```

(28) determines which edges should be rewired based on the probability `beta`.

(29) to (30) determines the new target nodes for the edges that are being rewired, ensuring that the new target is not the source node itself or any of its current neighbors.

(31) creates a graph `h` from the source nodes `s` and target nodes `t`.

(32) imports the txt file of each subject in the group.

(33) returns a group of subjects with connectivity data.

(34) assigns the panel element and don't confirm close.

Importer of a CON subject group from XLS/XLSX (ImporterGroupSubjectCON_XLS)

In this section we will show how to implement in detail `ImporterGroupSubjectCON_XLS`. The data should be stored in the folder 'Group1' and 'Group2', and the file format is '.xls' or '.xlsx'.

Code 10: **ImporterGroupSubjectCON_XLS element header.** The `header` section of the generator code for `_ImporterGroupSubjectCON_XLS.gen.m` provides the general information about the `Importer` element. ← Code 7

```
%% iheader!
ImporterGroupSubjectCON_XLS < Importer (im, importer of CON subject group
     from XLS/XLSX) imports a group of subjects with connectivity data from
     a series of XLS/XLSX file.

%%% idescription!
ImporterGroupSubjectCON_XLS imports a group of subjects with connectivity
     data from a series of XLS/XLSX files contained in a folder named "
     GROUP_ID". All these files must be in the same folder; also, no other
     files should be in the folder. Each file contains a table of values
     corresponding to the adjacency matrix. The variables of interest are
     from another XLS/XLSX file named "GROUP_ID.vois.xlsx" (if exisitng)
     consisting of the following columns: Subject ID (column 1), covariates
     (subsequent columns). The 1st row contains the headers, the 2nd row a
     string with the categorical variables of interest, and each subsequent
     row the values for each subject.

%%% iseealso!
Group, SubjectCON, ExporterGroupSubjectCON_XLS
```

Code 11: **ImporterGroupSubjectCON_XLS element prop update.** The `props_update` section of the generator code for `_ImporterGroupSubjectCON_XLS.gen.m` updates the properties of the `Importer` element. ← Code 8

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the CON subject group importer from
     XLS/XLSX.
%%%% idefault!
'ImporterGroupSubjectCON_XLS'

%%% iprop!
DESCRIPTION (constant, string) is the description of the CON subject group
     importer from XLS/XLSX.
%%%% idefault!
'ImporterGroupSubjectCON_XLS imports a group of subjects with connectivity
     data from a series of XLS/XLSX file. The variables of interest can be
     loaded from another XLS/XLSX file.'

%%% iprop!
TEMPLATE (parameter, item) is the template of the CON subject group importer
      from XLS/XLSX.
```

```
%%%% isettings!
'ImporterGroupSubjectCON_XLS'

%%% iprop!
ID (data, string) is a few-letter code for the CON subject group importer
     from XLS/XLSX.
%%%% idefault!
'ImporterGroupSubjectCON_XLS ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the CON subject group
     importer from XLS/XLSX.
%%%% idefault!
'ImporterGroupSubjectCON_XLS label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the CON subject group
      importer from XLS/XLSX.
%%%% idefault!
'ImporterGroupSubjectCON_XLS notes'

%% iprops!

%%% iprop!
DIRECTORY (data, string) is the directory containing the CON subject group
     files from which to load the subject group.
%%%% idefault!
fileparts(which('test_braph2'))

%%% iprop!
GET_DIR (query, item) opens a dialog box to set the directory from where to
     load the XLS/XLSX files of the CON subject group.
%%%% isettings!
'ImporterGroupSubjectCON_XLS'
%%%% icalculate!
directory = uigetdir('Select directory');
if ischar(directory) && isfolder(directory)
    im.set('DIRECTORY', directory);
end
value = im;

%%% iprop!
BA (data, item) is a brain atlas.
%%%% isettings!
'BrainAtlas'

%%% iprop!
GR (result, item) is a group of subjects with connectivity data.
%%%% isettings!
'Group'
%%%% icheck_value!
check = any(strcmp(value.get(Group.SUB_CLASS_TAG), subclasses('SubjectCON',
     [], [], true))); (1)
%%%% idefault!
Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', '
     SubjectCON')) (2)
%%%% icalculate! (3)
gr = Group( ...
    'SUB_CLASS', 'SubjectCON', ...
    'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON') ...
```

(1) Same as in note (3) of Code 8.

(2) Same as in note (4) of Code 8.

(3) Same as in note (5) to (28) in Code 8.

```
69      );
70
71  gr.lock('SUB_CLASS');
72
73  directory = im.get('DIRECTORY');
74  if isfolder(directory)
75      wb = braph2waitbar(im.get('WAITBAR'), 0, 'Reading directory ...');
76
77      [~, gr_name] = fileparts(directory);
78      gr.set( ...
79          'ID', gr_name, ...
80          'LABEL', gr_name, ...
81          'NOTES', ['Group loaded from ' directory] ...
82          );
83
84      try
85          braph2waitbar(wb, .15, 'Loading subject group ...')
86
87          % analyzes file
88          files = [dir(fullfile(directory, '*.xlsx')); dir(fullfile(directory,
        '*.xls'))];
89
90          if ~isempty(files)
91              % brain atlas
92              ba = im.get('BA');
93              if ba.get('BR_DICT').get('LENGTH') == 0
94                  % adds the number of regions of the first file to the brain
       atlas
95                  br_number = size(xlsread(fullfile(directory, files(1).name))
       , 1);
96                  br_dict = ba.memorize('BR_DICT');
97                  for j = 1:1:br_number
98                      br_dict.get('ADD', BrainRegion('ID', ['br' int2str(j)]))
99                  end
100             end
101
102             % adds subjects
103             sub_dict = gr.memorize('SUB_DICT');
104             for i = 1:1:length(files)
105                 braph2waitbar(wb, .15 + .85 * i / length(files), ['Loading
       subject ' num2str(i) ' of ' num2str(length(files)) ' ...'])
106
107                 % read file
108                 [~, sub_id] = fileparts(files(i).name);
109
110                 CON = xlsread(fullfile(directory, files(i).name));
111                 if size(CON, 1) ~= ba.get('BR_DICT').get('LENGTH') || size(
       CON, 2) ~= ba.get('BR_DICT').get('LENGTH')
112                     error( ...
113                         [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO], ...
114                         [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO '\\n'
       ...
115                         'The file ' sub_id ' should contain a matrix '
       int2str(ba.get('BR_DICT').get('LENGTH')) 'x' int2str(ba.get('BR_DICT').
       get('LENGTH')) ', ' ...
116                         'while it is ' int2str(size(CON, 1)) 'x' int2str(
       size(CON, 2)) '.'] ...
117                         )
118                 end
119
120                 sub = SubjectCON( ...
```

```
                    'ID', sub_id, ...
                    'BA', ba, ...
                    'CON', CON ...
                );
                sub_dict.get('ADD', sub);
            end

            % variables of interest
            vois = [];
            if isfile([directory '.vois.xls'])
                [~, ~, vois] = xlsread([directory '.vois.xls']);
            elseif isfile([directory '.vois.xlsx'])
                [~, ~, vois] = xlsread([directory '.vois.xlsx']);
            end
            if ~isempty(vois)
                for i = 3:1:size(vois, 1)
                    sub_id = vois{i, 1};
                    sub = sub_dict.get('IT', sub_id);
                    for v = 2:1:size(vois, 2)
                        voi_id = vois{1, v};
                        if isnumeric(vois{2, v}) % VOINumeric
                            sub.memorize('VOI_DICT').get('ADD', ...
                                VOINumeric( ...
                                    'ID', voi_id, ...
                                    'V', vois{i, v} ...
                                    ) ...
                                );
                        elseif ischar(vois{2, v}) % VOICategoric
                            sub.memorize('VOI_DICT').get('ADD', ...
                                VOICategoric( ...
                                    'ID', voi_id, ...
                                    'CATEGORIES', str2cell(vois{2, v}), ...
                                    'V', find(strcmp(vois{i, v}, str2cell(
    vois{2, v}))) ...
                                    ) ...
                                );
                        end
                    end
                end
            end
        end
    catch e
        braph2waitbar(wb, 'close')

        rethrow(e)
    end

    braph2waitbar(wb, 'close')
else
    error([BRAPH2.STR ':ImporterGroupSubjectCON_XLS:' BRAPH2.ERR_IO], ...
        [BRAPH2.STR ':ImporterGroupSubjectCON_XLS:' BRAPH2.ERR_IO '\\n' ...
        'The prop DIRECTORY must be an existing directory, but it is '''
    directory '''.'] ...
        );
end

value = gr;
```

Code 12: **ImporterGroupSubjectCON_XLS element tests. The `tests` section from the element generator** `_ImporterGroupSubjectCON_XLS.gen.m`. ← Code 8

```
%% itests!

%%% iexcluded_props!
[ImporterGroupSubjectCON_XLS.GET_DIR]

%%% itest!
%%%% iname!
Create example files
%%%% icode!
data_dir = [fileparts(which('SubjectCON')) filesep 'Example data CON XLS'];
if ~isdir(data_dir)
    mkdir(data_dir);

    % Brain Atlas
    im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx');
    ba = im_ba.get('BA');
    ex_ba = ExporterBrainAtlasXLS( ...
        'BA', ba, ...
        'FILE', [data_dir filesep() 'atlas.xlsx'] ...
        );
    ex_ba.get('SAVE')
    N = ba.get('BR_DICT').get('LENGTH');

    % saves RNG
    rng_settings_ = rng(); rng('default')

    sex_options = {'Female' 'Male'};

    % Group 1
    K1 = 2;
    beta1 = 0.3;
    gr1_name = 'CON_Group_1_XLS';
    gr1_dir = [data_dir filesep() gr1_name];
    mkdir(gr1_dir);
    vois1 = [
        {{'Subject ID'} {'Age'} {'Sex'}}
        {{} {} cell2str(sex_options)}
        ];
    for i = 1:1:50 % subject number
        sub_id = ['SubjectCON_' num2str(i)];

        h1 = WattsStrogatz(N, K1, beta1); % create two WS graph

        A1 = full(adjacency(h1)); A1(1:length(A1)+1:numel(A1)) = 0;
        r = 0 + (0.5 - 0)*rand(size(A1)); diffA = A1 - r; A1(A1 ~= 0) =
 diffA(A1 ~= 0);
        A1 = max(A1, transpose(A1)); % make the adjacency matrix symmetric

        writetable(array2table(A1), [gr1_dir filesep() sub_id '.xlsx'], '
 WriteVariableNames', false)
        vois1 = [vois1; {sub_id, randi(90), sex_options(randi(2))}];
    end
    writetable(table(vois1), [data_dir filesep() gr1_name '.vois.xlsx'], '
 WriteVariableNames', false)

    % Group 2
```

```
    K2 = 2;
    beta2 = 0.85;
    gr2_name = 'CON_Group_2_XLS';
    gr2_dir = [data_dir filesep() gr2_name];
    mkdir(gr2_dir);
    vois2 = [
        {{'Subject ID'} {'Age'} {'Sex'}}
        {{} {} cell2str(sex_options)}
        ];
    for i = 51:1:100
        sub_id = ['SubjectCON_' num2str(i)];

        h2 = WattsStrogatz(N, K2, beta2);
        % figure(2)
        % plot(h2, 'NodeColor',[1 0 0], 'EdgeColor',[0 0 0], 'EdgeAlpha
 ',0.1, 'Layout','circle');
        % title(['Group 2: Graph with $N = $ ' num2str(N_nodes) ...
        %     ' nodes, $K = $ ' num2str(K2) ', and $\beta = $ ' num2str(
 beta2)], ...
        %     'Interpreter','latex')
        % axis equal

        A2 = full(adjacency(h2)); A2(1:length(A2)+1:numel(A2)) = 0;
        r = 0 + (0.5 - 0)*rand(size(A2)); diffA = A2 - r; A2(A2 ~= 0) =
 diffA(A2 ~= 0);
        A2 = max(A2, transpose(A2));

        writetable(array2table(A2), [gr2_dir filesep() sub_id '.xlsx'], '
 WriteVariableNames', false)

        % variables of interest
        vois2 = [vois2; {sub_id, randi(90), sex_options(randi(2))}];
    end
    writetable(table(vois2), [data_dir filesep() gr2_name '.vois.xlsx'], '
 WriteVariableNames', false)

    % reset RNG
    rng(rng_settings_)
end

%%% itest_functions!
function h = WattsStrogatz(N,K,beta)
% H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
% nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
%
% beta = 0 is a ring lattice, and beta = 1 is a random graph.

% Connect each node to its K next and previous neighbors. This constructs
% indices for a ring lattice.
s = repelem((1:N)',1,K);
t = s + repmat(1:K,N,1);
t = mod(t-1,N)+1;

% Rewire the target node of each edge with probability beta
for source=1:N
    switchEdge = rand(K, 1) < beta;

    newTargets = rand(N, 1);
    newTargets(source) = 0;
    newTargets(s(t==source)) = 0;
    newTargets(t(source, ~switchEdge)) = 0;
```

```
111
112     [~, ind] = sort(newTargets, 'descend');
113     t(source, switchEdge) = ind(1:nnz(switchEdge));
114 end
115
116 h = graph(s,t);
117 end
118
119 %%% itest!
120 %%%% iname!
121 GUI
122 %%%% iprobability!
123 .01
124 %%%% icode!
125 im_ba = ImporterBrainAtlasXLS('FILE', [fileparts(which('SubjectCON'))
      filesep 'Example data CON XLS' filesep 'atlas.xlsx']);
126 ba = im_ba.get('BA');
127
128 im_gr = ImporterGroupSubjectCON_XLS( ...
129     'DIRECTORY', [fileparts(which('SubjectCON')) filesep 'Example data CON
      XLS' filesep 'CON_Group_1_XLS'], ...
130     'BA', ba, ...
131     'WAITBAR', true ...
132     );
133 gr = im_gr.get('GR');
134
135 gui = GUIElement('PE', gr, 'CLOSEREQ', false);
136 gui.get('DRAW')
137 gui.get('SHOW')
138
139 gui.get('CLOSE')
```

Exporter of a CON subject group from TXT (ExporterGroupSubjectCON_TXT)

In this section we will show how to implement in detail `ExporterGroupSubjectCON_TXT`. The data should be stored in the folder 'Group1' and 'Group2', and the file format is '.txt'.

Code 13: **ExporterGroupSubjectCON_TXT element header.** The `header` section of the generator code for `_ExporterGroupSubjectCON_TXT.gen.m` provides the general information about the `Exporter` element.

```
1
2 %% iheader!
3 ExporterGroupSubjectCON_TXT < Exporter (ex, exporter of CON subject group in
      TXT) exports a group of subjects with connectivity data to a series of
      TXT file. (1)
4
5 %%% idescription!
6 ExporterGroupSubjectCON_TXT exports a group of subjects with connectivity
     data to a series of tab-separated TXT files contained in a folder named
      "GROUP_ID". All these files are saved in the same folder. Each file
     contains a table of values corresponding to the adjacency matrix. The
     variables of interest (if existing) are saved in another tab-separated
     TXT file named "GROUP_ID.vois.txt" consisting of the following columns:
      Subject ID (column 1), covariates (subsequent columns). The 1st row
     contains the headers, the 2nd row a string with the categorical
     variables of interest, and each subsequent row the values for each
     subject.
7
8 %%% iseealso!
9 Group, SunbjectCON, ExporterGroupSubjectCON_TXT
```

(1) The element `ExporterGroupSubjectCON_TXT` is defined as a subclass of `Exporter`. The moniker will be `ex`.

Code 14: **ExporterGroupSubjectCON_TXT element prop update.** The `props_update` section of the generator code for `_ExporterGroupSubjectCON_TXT.gen.m` updates the properties of the `Exporter` element.

```
1
2 %% iprops_update!
3
4 %%% iprop!
5 NAME (constant, string) is the name of the CON subject group exporter in TXT
     .
6 %%%% idefault!
7 'ExporterGroupSubjectCON_TXT'
8
9 %%% iprop!
10 DESCRIPTION (constant, string) is the description of the CON subject group
     exporter in TXT.
11 %%%% idefault!
12 'ExporterGroupSubjectCON_TXT exports a group of subjects with connectivity
     data to a series of TXT file and their covariates age and sex (if
     existing) to another TXT file.'
13
14 %%% iprop!
15 TEMPLATE (parameter, item) is the template of the CON subject group exporter
      in TXT.
16 %%%% isettings!
```

```
'ExporterGroupSubjectCON_TXT'

%%% iprop!
ID (data, string) is a few-letter code for the CON subject group exporter in
      TXT.
%%%% idefault!
'ExporterGroupSubjectCON_TXT ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the CON subject group
     exporter in TXT.
%%%% idefault!
'ExporterGroupSubjectCON_TXT label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the CON subject group
      exporter in TXT.
%%%% idefault!
'ExporterGroupSubjectCON_TXT notes'

%% iprops!

%%% iprop!
GR (data, item) is a group of subjects with connectivity data.
%%%% isettings!
'Group'
%%%% icheck_value!
check = any(strcmp(value.get(Group.SUB_CLASS_TAG), subclasses('SubjectCON',
     [], [], true))); (1)
%%%% idefault!
Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', '
     SubjectCON'))

%%% iprop!
DIRECTORY (data, string) is the directory name where to save the group of
     subjects with connectivity data.
%%%% idefault!
[fileparts(which('test_braph2')) filesep '
     default_group_subjects_CON_most_likely_to_be_erased'] (2)

%%% iprop!
PUT_DIR (query, item) opens a dialog box to set the directory where to save
     the group of subjects with connectivity data.
%%%% isettings!
'ExporterGroupSubjectCON_TXT'
%%%% icalculate!
directory = uigetdir('Select directory'); (3)
if ischar(directory) && isfolder(directory) (4)
  ex.set('DIRECTORY', directory);
end
value = ex;

%%% iprop!
SAVE (result, empty) saves the group of subjects with connectivity data in
     TXT files in the selected directory.
%%%% icalculate!
directory = ex.get('DIRECTORY');

if isfolder(directory) (5)
    wb = braph2waitbar(ex.get('WAITBAR'), 0, 'Retrieving path ...'); (6)
```

(1) checks that the `SUB_CLASS_TAG` is equal to `'SubjectCON'`.

(2) defines the export directory.

(3) selects the export directory.

(4) checks that the export directory is correct.

(5) checks the export directory is a folder.

(6) creates the waitbar with an initial progress of `0`. Displaying the character `'Retrieving path ...'`.

```
68
69      gr = ex.get('GR');
70
71      gr_directory = [directory filesep() gr.get('ID')];
72      if ~exist(gr_directory, 'dir')
73          mkdir(gr_directory)
74      end
75
76      braph2waitbar(wb, .15, 'Organizing info ...')
77
78      sub_dict = gr.get('SUB_DICT');
79      sub_number = sub_dict.get('LENGTH');
80
81      for i = 1:1:sub_number
82          braph2waitbar(wb, .15 + .85 * i / sub_number, ['Saving subject '
         num2str(i) ' of ' num2str(sub_number) '...']) (7)
83
84          sub = sub_dict.get('IT', i); (8)
85          sub_id = sub.get('ID'); (9)
86          sub_CON = sub.get('CON'); (10)
87
88          tab = table(sub_CON); (11)
89
90          sub_file = [gr_directory filesep() sub_id '.txt'];
91
92          % save file
93          writetable(tab, sub_file, 'Delimiter', '\t', 'WriteVariableNames',
         false); (12)
94      end
95
96      % variables of interest
97      voi_ids = {};
98      for i = 1:1:sub_number
99          sub = sub_dict.get('IT', i);
100         voi_ids = unique([voi_ids, sub.get('VOI_DICT').get('KEYS')]); (13)
101     end
102     if ~isempty(voi_ids)
103         vois = cell(2 + sub_number, 1 + length(voi_ids));
104         vois{1, 1} = 'Subject ID';
105         vois(1, 2:end) = voi_ids;
106         for i = 1:1:sub_number
107             sub = sub_dict.get('IT', i);
108             vois{2 + i, 1} = sub.get('ID');
109
110             voi_dict = sub.get('VOI_DICT');
111             for v = 1:1:voi_dict.get('LENGTH') (14)
112                 voi = voi_dict.get('IT', v);
113                 voi_id = voi.get('ID');
114                 if isa(voi, 'VOINumeric') % Numeric
115                     vois{2 + i, 1 + find(strcmp(voi_id, voi_ids))} = voi.get
         ('V');
116                 elseif isa(voi, 'VOICategoric') % Categoric
117                     categories = voi.get('CATEGORIES');
118                     vois{2, 1 + find(strcmp(voi_id, voi_ids))} = {['{'
         sprintf(' ''%s'' ', categories{:}) '}']};
119                     vois{2 + i, 1 + find(strcmp(voi_id, voi_ids))} =
         categories{voi.get('V')};
120                 end
```

(7) updates the waitbar.

(8) extracts the information of one subject.

(9) extracts the 'ID' of the subject.

(10) extracts the 'CON' of the subject.

(11) changes the matrix to type of table.

(12) writes the table to txt file.

(13) extracts the keys of the variables of interest.

(14) saves the value of each variable of interest.

```
121             end
122         end
123         writetable(table(vois), [gr_directory '.vois.txt'], 'Delimiter', '\t
        ', 'WriteVariableNames', false) (15)
124     end
125
126     braph2waitbar(wb, 'close') (16)
127 end
128 value = [];
```

(15) writes the table of variable of interest to txt file.

(16) closes the waitbar.

Code 15: **ExporterGroupSubjectCON_TXT element tests.** The `tests` section from the element generator _ExporterGroupSubjectCON_TXT.gen.m.

```
%% itests!

%%% iexcluded_props! (1)
[ExporterGroupSubjectCON_TXT.PUT_DIR]

%%% itest!
%%%% iname!
Delete directory TBE (2)
%%%% iprobability!
1
%%%% icode!
warning('off', 'MATLAB:DELETE:FileNotFound')
dir_to_be_erased = ExporterGroupSubjectCON_TXT.getPropDefault('DIRECTORY');
if isfolder(dir_to_be_erased)
    rmdir(dir_to_be_erased, 's')
end
warning('on', 'MATLAB:DELETE:FileNotFound')

%%% itest!
%%%% iname!
Export and import (3)
%%%% iprobability!
.01
%%%% icode!
br1 = BrainRegion( ... (4)
    'ID', 'ISF', ...
    'LABEL', 'superiorfrontal', ...
    'NOTES', 'notes1', ...
    'X', -12.6, ...
    'Y', 22.9, ...
    'Z', 42.4 ...
    );
br2 = BrainRegion( ...
    'ID', 'lFP', ...
    'LABEL', 'frontalpole', ...
    'NOTES', 'notes2', ...
    'X', -8.6, ...
    'Y', 61.7, ...
    'Z', -8.7 ...
    );
br3 = BrainRegion( ...
    'ID', 'lRMF', ...
    'LABEL', 'rostralmiddlefrontal', ...
    'NOTES', 'notes3', ...
    'X', -31.3, ...
    'Y', 41.2, ...
    'Z', 16.5 ...
    );
br4 = BrainRegion( ...
    'ID', 'lCMF', ...
    'LABEL', 'caudalmiddlefrontal', ...
    'NOTES', 'notes4', ...
    'X', -34.6, ...
    'Y', 10.2, ...
    'Z', 42.8 ...
```

(1) List of properties that are excluded from testing.

(2) deletes the example files.

(3) tests importer and exporter functions.

(4) creates the `BrainRegion` with `'ID'`, `'LABEL'`, `'NOTES'`, `'X'`, `'Y'`, and `'Z'`.

```
    );
br5 = BrainRegion( ...
    'ID', 'lPOB', ...
    'LABEL', 'parsorbitalis', ...
    'NOTES', 'notes5', ...
    'X', -41, ...
    'Y', 38.8, ...
    'Z', -11.1 ...
    );

ba = BrainAtlas( ...
    'ID', 'TestToSaveCoolID', ...
    'LABEL', 'Brain Atlas', ...
    'NOTES', 'Brain atlas notes', ...
    'BR_DICT', IndexedDictionary('IT_CLASS', 'BrainRegion', 'IT_LIST', {br1,
     br2, br3, br4, br5}) ... (5)
    );

sub1 = SubjectCON( ... (6)
    'ID', 'SUB CON 1', ...
    'LABEL', 'Subejct CON 1', ...
    'NOTES', 'Notes on subject CON 1', ...
    'BA', ba, ...
    'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
    );
sub1.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 75)) (7)
sub1.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES',
     {'Female', 'Male'}, 'V', find(strcmp('Female', {'Female', 'Male'}))))
     (8)

sub2 = SubjectCON( ...
    'ID', 'SUB CON 2', ...
    'LABEL', 'Subejct CON 2', ...
    'NOTES', 'Notes on subject CON 2', ...
    'BA', ba, ...
    'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
    );
sub2.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 70))
sub2.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES',
     {'Female', 'Male'}, 'V', find(strcmp('Male', {'Female', 'Male'}))))

sub3 = SubjectCON( ...
    'ID', 'SUB CON 3', ...
    'LABEL', 'Subejct CON 3', ...
    'NOTES', 'Notes on subject CON 3', ...
    'BA', ba, ...
    'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
    );
sub3.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 50))
sub3.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES',
     {'Female', 'Male'}, 'V', find(strcmp('Female', {'Female', 'Male'}))))

gr = Group( ...
    'ID', 'GR CON', ...
    'LABEL', 'Group label', ...
    'NOTES', 'Group notes', ...
    'SUB_CLASS', 'SubjectCON', ...
    'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON', 'IT_LIST', {sub1
     , sub2, sub3}) ... (9)
    );
```

(5) merges the 5 created brain regions as the `BrainAtlas`.

(6) creates the `SubjectCON` with `'ID'`, `'LABEL'`, `'NOTES'`, `'BA'`, and `'CON'`.

(7) adds the variables of interest `'Age'`.

(8) adds the variables of interest `'Sex'`.

(9) merges the 3 created subjects as the `Group`.

```
111
112 directory = [fileparts(which('test_braph2')) filesep '
        trial_group_subjects_CON_to_be_erased']; (10)
113 if ~exist(directory, 'dir')
114     mkdir(directory)
115 end
116
117 ex = ExporterGroupSubjectCON_TXT( ... (11)
118     'DIRECTORY', directory, ...
119     'GR', gr ...
120     );
121 ex.get('SAVE');
122
123 im1 = ImporterGroupSubjectCON_TXT( ... (12)
124     'DIRECTORY', [directory filesep() gr.get(Group.ID)], ...
125     'BA', ba ...
126     );
127 gr_loaded1 = im1.get('GR');
128
129 assert(gr.get('SUB_DICT').get('LENGTH') == gr_loaded1.get('SUB_DICT').get('
        LENGTH'), ...
130   [BRAPH2.STR ':ExporterGroupSubjectCON_TXT:' BRAPH2.FAIL_TEST], ...
131     'Problems saving or loading a group.') (13)
132 for i = 1:1:max(gr.get('SUB_DICT').get('LENGTH'), gr_loaded1.get('SUB_DICT')
        .get('LENGTH')) (14)
133     sub = gr.get('SUB_DICT').get('IT', i);
134     sub_loaded = gr_loaded1.get('SUB_DICT').get('IT', i);
135     assert( ...(15)
136         isequal(sub.get('ID'), sub_loaded.get('ID')) & ...
137         isequal(sub.get('BA'), sub_loaded.get('BA')) & ...
138         isequal(sub.get('VOI_DICT').get('IT', 'Age').get('V'), sub_loaded.
        get('VOI_DICT').get('IT', 'Age').get('V')) & ...
139         isequal(sub.get('VOI_DICT').get('IT', 'Sex').get('V'), sub_loaded.
        get('VOI_DICT').get('IT', 'Sex').get('V')) & ...
140         isequal(round(sub.get('CON'), 10), round(sub_loaded.get('CON'), 10))
        , ...
141         [BRAPH2.STR ':ExporterGroupSubjectCON_TXT:' BRAPH2.FAIL_TEST], ...
142         'Problems saving or loading a group.')
143 end
144
145
146 im2 = ImporterGroupSubjectCON_TXT( ... (16)
147     'DIRECTORY', [directory filesep() gr.get(Group.ID)] ...
148     );
149 gr_loaded2 = im2.get('GR');
150
151 assert(gr.get('SUB_DICT').get('LENGTH') == gr_loaded2.get('SUB_DICT').get('
        LENGTH'), ...
152   [BRAPH2.STR ':ExporterGroupSubjectCON_TXT:' BRAPH2.FAIL_TEST], ...
153     'Problems saving or loading a group.')
154 for i = 1:1:max(gr.get('SUB_DICT').get('LENGTH'), gr_loaded2.get('SUB_DICT')
        .get('LENGTH'))
155     sub = gr.get('SUB_DICT').get('IT', i);
156     sub_loaded = gr_loaded2.get('SUB_DICT').get('IT', i);
157     assert( ...
158         isequal(sub.get('ID'), sub_loaded.get('ID')) & ...
159         ~isequal(sub.get('BA').get('ID'), sub_loaded.get('BA').get('ID')) &
```

(10) defines the directory of data.

(11) exports the txt files of data.

(12) imports the txt files of data.

(13) checks the size of data is same and get the wrong information if having.

(14) checks each property is same.

(15) checks the properties 'ID', 'BA', 'Age', 'Sex' and 'CON',] between loaded data and saved data are same.

(16) checks the data in group 2. Same as note in (12) to (15)

```
    ...
160         isequal(sub.get('VOI_DICT').get('IT', 'Age').get('V'), sub_loaded.
    get('VOI_DICT').get('IT', 'Age').get('V')) & ...
161         isequal(sub.get('VOI_DICT').get('IT', 'Sex').get('V'), sub_loaded.
    get('VOI_DICT').get('IT', 'Sex').get('V')) & ...
162         isequal(round(sub.get('CON'), 10), round(sub_loaded.get('CON'), 10))
    , ...
163         [BRAPH2.STR ':ExporterGroupSubjectCON_TXT:' BRAPH2.FAIL_TEST], ...
164         'Problems saving or loading a group.')
165 end
166
167 rmdir(directory, 's') (17)
```

(17) deletes the testing data.

Exporter of a CON subject group from XLS/XLSX (ExporterGroupSubjectCON_XLS)

In this section we will show how to implement in detail `ExporterGroupSubjectCON_XLS`. The data should be stored in the folder 'Group1' and 'Group2', and the file format is '.txt'.

Code 16: **ExporterGroupSubjectCON_XLS element header.** The `header` section of the generator code for `_ExporterGroupSubjectCON_XLS.gen.m` provides the general information about the `Exporter` element. ← Code 13

```
1
2 %% !header!
3 ExporterGroupSubjectCON_XLS < Exporter (ex, exporter of CON subject group in
      XLSX) exports a group of subjects with connectivity data to a series
     of XLSX file.
4
5 %%% !description!
6 ExporterGroupSubjectCON_XLS exports a group of subjects with connectivity
     data to a series of XLSX files contained in a folder named "GROUP_ID".
     All these files are saved in the same folder. Each file contains a
     table of values corresponding to the adjacency matrix.The variables of
     interest (if existing) are saved in another XLSX file  named "GROUP_ID.
     vois.xlsx" consisting of the following columns: Subject ID (column 1),
     covariates (subsequent columns). The 1st row contains the headers, the
     2nd row a string with the categorical variables of interest, and each
     subsequent row the values for each subject.
7
8 %%% !seealso!
9 Group, SunbjectCON, ImporterGroupSubjectCON_XLS
```

Code 17: **ExporterGroupSubjectCON_XLS element prop update.** The `props_update` section of the generator code for `_ExporterGroupSubjectCON_XLS.gen.m` updates the properties of the `Exporter` element. ← Code **??**

```
1
2 %% !props_update!
3
4 %%% !prop!
5 NAME (constant, string) is the name of the CON subject group exporter in
     XLSX.
6 %%%% !default!
7 'ExporterGroupSubjectCON_XLS'
8
9 %%% !prop!
10 DESCRIPTION (constant, string) is the description of the CON subject group
      exporter in XLSX.
11 %%%% !default!
12 'ExporterGroupSubjectCON_XLS exports a group of subjects with connectivity
     data to a series of XLSX files. The variables of interest (if existing)
      are saved in another XLSX file.'
13
14 %%% !prop!
15 TEMPLATE (parameter, item) is the template of the CON subject group exporter
      in XLSX.
```

```
%%%% isettings!
'ExporterGroupSubjectCON_XLS'

%%% iprop!
ID (data, string) is a few-letter code for the CON subject group exporter in XLSX.
%%%% idefault!
'ExporterGroupSubjectCON_XLS ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the CON subject group exporter in XLSX.
%%%% idefault!
'ExporterGroupSubjectCON_XLS label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the CON subject group exporter in XLSX.
%%%% idefault!
'ExporterGroupSubjectCON_XLS notes'

%% iprops!

%%% iprop!
GR (data, item) is a group of subjects with connectivity data.
%%%% isettings!
'Group'
%%%% icheck_value!
check = any(strcmp(value.get(Group.SUB_CLASS_TAG), subclasses('SubjectCON', [], [], true))); % Format.checkFormat(Format.ITEM, value, 'Group') already checked
%%%% idefault! (1)
Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON'))

%%% iprop!
DIRECTORY (data, string) is the directory name where to save the group of subjects with connectivity data.
%%%% idefault!
[fileparts(which('test_braph2')) filesep 'default_group_subjects_CON_most_likely_to_be_erased']

%%% iprop!
PUT_DIR (query, item) opens a dialog box to set the directory where to save the group of subjects with connectivity data.
%%%% isettings!
'ExporterGroupSubjectCON_XLS'
%%%% icalculate!
directory = uigetdir('Select directory');
if ischar(directory) && isfolder(directory)
    ex.set('DIRECTORY', directory);
end
value = ex;

%%% iprop!
SAVE (result, empty) saves the group of subjects with connectivity data in XLSX files in the selected directory.
%%%% icalculate!
directory = ex.get('DIRECTORY');

if isfolder(directory) (2)
```

(1) Same as in note (1) of Code 14.

(2) Same as in note (4) to (17) in Code 14.

```
    wb = braph2waitbar(ex.get('WAITBAR'), 0, 'Retrieving path ...');

    gr = ex.get('GR');

    gr_directory = [directory filesep() gr.get('ID')];
    if ~exist(gr_directory, 'dir')
        mkdir(gr_directory)
    end

braph2waitbar(wb, .15, 'Organizing info ...')

    sub_dict = gr.get('SUB_DICT');
    sub_number = sub_dict.get('LENGTH');

    for i = 1:1:sub_number
        braph2waitbar(wb, .15 + .85 * i / sub_number, ['Saving subject '
 num2str(i) ' of ' num2str(sub_number) ' ...'])

        sub = sub_dict.get('IT', i);
        sub_id = sub.get('ID');
        sub_CON = sub.get('CON');

        tab = table(sub_CON);

        sub_file = [gr_directory filesep() sub_id '.xlsx'];

        % save file
        writetable(tab, sub_file, 'WriteVariableNames', false);
    end

    % variables of interest
    voi_ids = {};
    for i = 1:1:sub_number
        sub = sub_dict.get('IT', i);
        voi_ids = unique([voi_ids, sub.get('VOI_DICT').get('KEYS')]);
    end
    if ~isempty(voi_ids)
        vois = cell(2 + sub_number, 1 + length(voi_ids));
        vois{1, 1} = 'Subject ID';
        vois(1, 2:end) = voi_ids;
        for i = 1:1:sub_number
            sub = sub_dict.get('IT', i);
            vois{2 + i, 1} = sub.get('ID');

            voi_dict = sub.get('VOI_DICT');
            for v = 1:1:voi_dict.get('LENGTH')
                voi = voi_dict.get('IT', v);
                voi_id = voi.get('ID');
                if isa(voi, 'VOINumeric') % Numeric
                    vois{2 + i, 1 + find(strcmp(voi_id, voi_ids))} = voi.get
 ('V');
                elseif isa(voi, 'VOICategoric') % Categoric
                    categories = voi.get('CATEGORIES');
                    vois{2, 1 + find(strcmp(voi_id, voi_ids))} = cell2str(
 categories);
                    vois{2 + i, 1 + find(strcmp(voi_id, voi_ids))} =
 categories{voi.get('V')};
                end
            end
        end
        writetable(table(vois), [gr_directory '.vois.xlsx'], '
```

```
      WriteVariableNames', false)
124     end
125
126     braph2waitbar(wb, 'close')
127 end
128
129 value = [];
```

Code 18: **ExporterGroupSubjectCON_XLS element tests.** The `tests` section from the element generator _ExporterGroupSubject-CON_XLS.gen.m. ← Code 15

```
%% itests!

%%% iexcluded_props!
[ExporterGroupSubjectCON_XLS.PUT_DIR]

%%% itest!
%%%% iname!
Delete directory TBE
%%%% iprobability!
1
%%%% icode!
warning('off', 'MATLAB:DELETE:FileNotFound')
dir_to_be_erased = ExporterGroupSubjectCON_XLS.getPropDefault('DIRECTORY');
if isfolder(dir_to_be_erased)
    rmdir(dir_to_be_erased, 's')
end
warning('on', 'MATLAB:DELETE:FileNotFound')

%%% itest!
%%%% iname!
Export and import
%%%% iprobability!
.01
%%%% icode!
br1 = BrainRegion( ...
    'ID', 'ISF', ...
    'LABEL', 'superiorfrontal', ...
    'NOTES', 'notes1', ...
    'X', -12.6, ...
    'Y', 22.9, ...
    'Z', 42.4 ...
    );
br2 = BrainRegion( ...
    'ID', 'lFP', ...
    'LABEL', 'frontalpole', ...
    'NOTES', 'notes2', ...
    'X', -8.6, ...
    'Y', 61.7, ...
    'Z', -8.7 ...
    );
br3 = BrainRegion( ...
    'ID', 'lRMF', ...
    'LABEL', 'rostralmiddlefrontal', ...
    'NOTES', 'notes3', ...
    'X', -31.3, ...
    'Y', 41.2, ...
    'Z', 16.5 ...
    );
br4 = BrainRegion( ...
    'ID', 'lCMF', ...
    'LABEL', 'caudalmiddlefrontal', ...
    'NOTES', 'notes4', ...
    'X', -34.6, ...
    'Y', 10.2, ...
    'Z', 42.8 ...
    );
br5 = BrainRegion( ...
```

```
    'ID', 'lPOB', ...
    'LABEL', 'parsorbitalis', ...
    'NOTES', 'notes5', ...
    'X', -41, ...
    'Y', 38.8, ...
    'Z', -11.1 ...
    );

ba = BrainAtlas( ...
    'ID', 'TestToSaveCoolID', ...
    'LABEL', 'Brain Atlas', ...
    'NOTES', 'Brain atlas notes', ...
    'BR_DICT', IndexedDictionary('IT_CLASS', 'BrainRegion', 'IT_LIST', {br1,
     br2, br3, br4, br5}) ...
    );

sub1 = SubjectCON( ...
    'ID', 'SUB CON 1', ...
    'LABEL', 'Subejct CON 1', ...
    'NOTES', 'Notes on subject CON 1', ...
    'BA', ba, ...
    'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
    );
sub1.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 75))
sub1.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES',
     {'Female', 'Male'}, 'V', find(strcmp('Female', {'Female', 'Male'}))))

sub2 = SubjectCON( ...
    'ID', 'SUB CON 2', ...
    'LABEL', 'Subejct CON 2', ...
    'NOTES', 'Notes on subject CON 2', ...
    'BA', ba, ...
    'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
    );
sub2.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 70))
sub2.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES',
     {'Female', 'Male'}, 'V', find(strcmp('Male', {'Female', 'Male'}))))

sub3 = SubjectCON( ...
    'ID', 'SUB CON 3', ...
    'LABEL', 'Subejct CON 3', ...
    'NOTES', 'Notes on subject CON 3', ...
    'BA', ba, ...
    'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
    );
sub3.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 50))
sub3.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES',
     {'Female', 'Male'}, 'V', find(strcmp('Female', {'Female', 'Male'}))))

gr = Group( ...
    'ID', 'GR CON', ...
    'LABEL', 'Group label', ...
    'NOTES', 'Group notes', ...
    'SUB_CLASS', 'SubjectCON', ...
    'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON', 'IT_LIST', {sub1
     , sub2, sub3}) ...
    );

directory = [fileparts(which('test_braph2')) filesep '
     trial_group_subjects_CON_to_be_erased'];
if ~exist(directory, 'dir')
```

```
    mkdir(directory)
end

ex = ExporterGroupSubjectCON_XLS( ...
    'DIRECTORY', directory, ...
    'GR', gr ...
    );
ex.get('SAVE');

im1 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [directory filesep() gr.get(Group.ID)], ...
    'BA', ba ...
    );
gr_loaded1 = im1.get('GR');

assert(gr.get('SUB_DICT').get('LENGTH') == gr_loaded1.get('SUB_DICT').get('
     LENGTH'), ...
  [BRAPH2.STR ':ExporterGroupSubjectCON_XLS:' BRAPH2.FAIL_TEST], ...
    'Problems saving or loading a group.')
for i = 1:1:max(gr.get('SUB_DICT').get('LENGTH'), gr_loaded1.get('SUB_DICT')
     .get('LENGTH'))
    sub = gr.get('SUB_DICT').get('IT', i);
    sub_loaded = gr_loaded1.get('SUB_DICT').get('IT', i);
    assert( ...
        isequal(sub.get('ID'), sub_loaded.get('ID')) & ...
        isequal(sub.get('BA'), sub_loaded.get('BA')) & ...
        isequal(sub.get('VOI_DICT').get('IT', 'Age').get('V'), sub_loaded.
     get('VOI_DICT').get('IT', 'Age').get('V')) & ...
        isequal(sub.get('VOI_DICT').get('IT', 'Sex').get('V'), sub_loaded.
     get('VOI_DICT').get('IT', 'Sex').get('V')) & ...
        isequal(sub.get('CON'), sub_loaded.get('CON')), ...
        [BRAPH2.STR ':ExporterGroupSubjectCON_XLS:' BRAPH2.FAIL_TEST], ...
        'Problems saving or loading a group.')
end

% import with new brain atlas
im2 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [directory filesep() gr.get(Group.ID)] ...
    );
gr_loaded2 = im2.get('GR');

assert(gr.get('SUB_DICT').get('LENGTH') == gr_loaded2.get('SUB_DICT').get('
     LENGTH'), ...
  [BRAPH2.STR ':ExporterGroupSubjectCON_XLS:' BRAPH2.FAIL_TEST], ...
    'Problems saving or loading a group.')
for i = 1:1:max(gr.get('SUB_DICT').get('LENGTH'), gr_loaded2.get('SUB_DICT')
     .get('LENGTH'))
    sub = gr.get('SUB_DICT').get('IT', i);
    sub_loaded = gr_loaded2.get('SUB_DICT').get('IT', i);
    assert( ...
        isequal(sub.get('ID'), sub_loaded.get('ID')) & ...
        ~isequal(sub.get('BA').get('ID'), sub_loaded.get('BA').get('ID')) &
     ...
        isequal(sub.get('VOI_DICT').get('IT', 'Age').get('V'), sub_loaded.
     get('VOI_DICT').get('IT', 'Age').get('V')) & ...
        isequal(sub.get('VOI_DICT').get('IT', 'Sex').get('V'), sub_loaded.
     get('VOI_DICT').get('IT', 'Sex').get('V')) & ...
        isequal(sub.get('CON'), sub_loaded.get('CON')), ...
        [BRAPH2.STR ':ExporterGroupSubjectCON_XLS:' BRAPH2.FAIL_TEST], ...
        'Problems saving or loading a group.')
end
```

```
165
166 rmdir(directory, 's')
```

Implementation of a subject with functional data

Subject with functional data (SubjectFUN)

In this section we will show how to implement in detail `SubjectFUN`. The connectivity matrix can be obtained from fMRI data.

Code 19: **SubjectFUN element header.** The `header` section of the generator code for `_SubjectFUN.gen.m` provides the general information about the `SubjectFUN` element.← Code 1

```
%% !header!
SubjectFUN < Subject (sub, subject with functional matrix) is a subject with
      functional matrix (e.g. fMRI).

%%% !description!
Subject with a functional matrix (e.g. obtained from fMRI).

%%% !seealso!
ImporterGroupSubjectFUN_TXT, ExporterGroupSubjectFUN_TXT,
     ImporterGroupSubjectFUN_XLS, ExporterGroupSubjectFUN_XLS
```

Code 20: **SubjectFUN element prop update.** The `props_update` section of the generator code for `_SubjectFUN.gen.m` updates the properties of the `Subject` element. ← Code 2

```
%% !props_update!

%%% !prop!
NAME (constant, string) is the name of the subject.
%%%% !default!
'SubjectFUN'

%%% !prop!
DESCRIPTION (constant, string) is the description of the subject.
%%%% !default!
'Subject with a functional matrix (e.g. obtained from fMRI).'

%%% !prop!
TEMPLATE (parameter, item) is the template of the subject.
%%% !settings!
'SubjectFUN'

%%% !prop!
ID (data, string) is a few-letter code for the subject.
%%%% !default!
'SubjectFUN ID'

%%% !prop!
LABEL (metadata, string) is an extended label of the subject.
%%%% !default!
'SubjectFUN label'

%%% !prop!
NOTES (metadata, string) are some specific notes about the subject.
%%%% !default!
'SubjectFUN notes'
```

```
32
33 %% iprops!
34
35 %%% iprop!
36 BA (data, item) is a brain atlas.
37 %%%% isettings!
38 'BrainAtlas'
39
40 %%% iprop!
41 FUN (data, matrix) is an adjacency matrix.
42 %%%% icheck_value!
43 br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
44 check = size(value, 2) == br_number; (1)
45 if check
46     msg = 'All ok!';
47 else
48     msg = ['FUN must be a matrix with the same number of columns as the
       brain regions (' int2str(br_number) ').'];
49 end
50 %%%% igui! (2)
51 pr = PanelPropMatrix('EL', sub, 'PROP', SubjectFUN.FUN, ...
52     'ROWNAME', {'numbered'}, ...
53     'COLUMNNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
54     varargin{:});
```

(1) checks the size of the column of `value` is equal to the number of brain regions. The rows of `value` represent the time series.

(2) Same as in note (4) of Code 2.

Code 21: **SubjectFUN element tests.** The `tests` section from the element generator `_SubjectFUN.gen.m`. ← Code 3

```
%% itests!

%%% itest!
%%%% iname!
GUI
%%%% iprobability!
.01
%%%% icode!
im_ba = ImporterBrainAtlasXLS('FILE', 'aal90_atlas.xlsx');
ba = im_ba.get('BA');

gr = Group('SUB_CLASS', 'SubjectFUN', 'SUB_DICT', IndexedDictionary('
     IT_CLASS', 'SubjectFUN'));
for i = 1:1:50
    sub = SubjectFUN( ...
        'ID', ['SUB FUN ' int2str(i)], ...
        'LABEL', ['Subejct FUN ' int2str(i)], ...
        'NOTES', ['Notes on subject FUN ' int2str(i)], ...
        'BA', ba, ...
        'FUN', rand(10, ba.get('BR_DICT').get('LENGTH')) ...(1)
        );
    sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
     rand()))
    sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
     CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
    gr.get('SUB_DICT').get('ADD', sub)
end

gui = GUIElement('PE', gr, 'CLOSEREQ', false);
gui.get('DRAW')
gui.get('SHOW')

gui.get('CLOSE')
```

(1) constructs the random adjacency matrix with the size of 10 timepoints by the number of brain regions.

Subject with functional multiplex data (SubjectFUN_MP)

In this section we will show how to implement in detail `SubjectFUN_MP`. The functional matrix can be obtained from fMRI data.

Code 22: **SubjectFUN_MP element header.** The `header` section of the generator code for `_SubjectFUN_MP.gen.m` provides the general information about the `SubjectFUN_MP` element. ← Code 4

```

%% iheader!
SubjectFUN_MP < Subject (sub, subject with functional multiplex data) is a
     subject with functional multiplex data (e.g. multiplex fMRI).

%%% idescription!
Subject with data for each brain region corresponding to L functional layers
      (e.g. activation timeseries obtaiend from fMRI or EEG).

%%% iseealso!
ImporterGroupSubjectFUN_MP_TXT, ExporterGroupSubjectFUN_MP_TXT,
     ImporterGroupSubjectFUN_MP_XLS, ExporterGroupSubjectFUN_MP_XLS
```

Code 23: **SubjectFUN_MP element prop update.** The `props_update` section of the generator code for `_SubjectFUN_MP.gen.m` updates the properties of the `Subject` element. ← Code 5

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the subject.
%%%% idefault!
'SubjectFUN_MP'

%%% iprop!
DESCRIPTION (constant, string) is the description of the subject.
%%%% idefault!
'Subject with data for each brain region corresponding to L functional
     layers (e.g. activation timeseries obtaiend from fMRI or EEG).'

%%% iprop!
TEMPLATE (parameter, item) is the template of the subject.
%%% isettings!
'SubjectFUN_MP'

%%% iprop!
ID (data, string) is a few-letter code for the subject.
%%%% idefault!
'SubjectFUN_MP ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the subject.
%%%% idefault!
'SubjectFUN_MP label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the subject.
%%%% idefault!
'SubjectFUN_MP notes'

```

```
%% iprops!

%%% iprop!
BA (data, item) is a brain atlas.
%%%% isettings!
'BrainAtlas'

%%% iprop!
L (data, scalar) is the number of layers of subject data. (1)
%%%% idefault!
2

%%% iprop!
LAYERLABELS (metadata, stringlist) are the layer labels provided by the user
    . (2)

%%% iprop!
ALAYERLABELS (query, stringlist) returns the processed layer labels. (3)
%%%% icalculate!
value = sub.get('LAYERLABELS');

%%% iprop!
FUN_MP (data, cell) is a cell containing L matrices with each column
    corresponding to the time series of a brain region.
%%%% icheck_value!
br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
num_layers = sub.get('L');
check = (iscell(value) && isequal(length(value), num_layers)  && isequal(
    cellfun(@(v) size(v, 2), value), ones(1, num_layers) * br_number)) || (
    isempty(value) && br_number == 0);  (4)
if check
    msg = 'All ok!';
else
    msg = ['FUN_MP must be a cell with L matrices with the same number of
    columns as the number of brain regions (' int2str(br_number) ').'];
end
%%%% igui! (5)
pr = PanelPropCell('EL', sub, 'PROP', SubjectFUN_MP.FUN_MP, ...
    'TABLE_HEIGHT', s(40), ...
    'XSLIDERSHOW', true, ...
    'XSLIDERLABELS', sub.getCallback('ALAYERLABELS'), ...
    'YSLIDERSHOW', false, ...
    'ROWNAME', {'numbered'}, ...
    'COLUMNNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
    varargin{:});
```

(1) Same as in note (1) of Code 5.

(2) Same as in note (2) of Code 5.

(3) Same as in note (3) of Code 5.

(4) checks the size of each layer are equal to the number of brain regions. The size of each layer is the length of time series by the number of regions.

(5) Same as in note (8) (9) (10) (11) of Code 5.

Code 24: **SubjectFUN_MP element tests.** The `tests` section from the element generator `_SubjectFUN_MP.gen.m`. ← Code 6

```
%% itests!

%%% itest!
%%%% iname!
GUI
%%%% iprobability!
.01
%%%% icode!
im_ba = ImporterBrainAtlasXLS('FILE', 'aal90_atlas.xlsx');
ba = im_ba.get('BA');
```

```
11
12 gr = Group('SUB_CLASS', 'SubjectFUN_MP', 'SUB_DICT', IndexedDictionary('
       IT_CLASS', 'SubjectFUN_MP'));
13 for i = 1:1:10 (1)
14     sub = SubjectFUN_MP( ...
15         'ID', ['SUB FUN_MP ' int2str(i)], ...
16         'LABEL', ['Subejct FUN_MP ' int2str(i)], ...
17         'NOTES', ['Notes on subject FUN_MP ' int2str(i)], ...
18         'BA', ba, ...
19         'L', 3, ...
20         'LAYERLABELS', {'L1' 'L2' 'L3'}, ...
21         'FUN_MP', {rand(10, ba.get('BR_DICT').get('LENGTH')), rand(10, ba.
       get('BR_DICT').get('LENGTH')), rand(10, ba.get('BR_DICT').get('LENGTH')
       )} ...
22         );
23     sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
       rand()))
24     sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
       CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
25     gr.get('SUB_DICT').get('ADD', sub)
26 end
27
28 gui = GUIElement('PE', gr, 'CLOSEREQ', false);
29 gui.get('DRAW')
30 gui.get('SHOW')
31
32 gui.get('CLOSE')
```

(1) Same as in note (1) (2) (3) of Code 6.

Implementation of a subject with connectivity and functional data

Subject with connectivity and functional multiplex data (SubjectCON_FUN_MP)

In this section we will show how to implement detail `SubjectCON_FUN_MP`. The connectivity matrix can be obtained from DTI data and the functional matrix can be obtained from fMRI data.

Code 25: **SubjectCON_FUN_MP element header.** The `header` section of the generator code for `_SubjectCON_FUN_MP.gen.m` provides the general information about the `SubjectCON_FUN_MP` element. ← Code 4

```
%% iheader!
SubjectCON_FUN_MP < Subject (sub, subject with connectivity and functional
     multiplex data) is a subject with connectivity and functional multiplex
      data (e.g. DTI and fMRI).

%%% idescription!
Subject with connectivity and functional data (e.g. obtained from DTI and
     fMRI).
The first layer contains a connectivity matrix and the second layer contains
      functional data.

%%% iseealso!
CombineGroups_CON_FUN_MP, SeparateGroups_CON_FUN_MP
```

Code 26: **SubjectCON_FUN_MP element prop update.** The `props_update` section of the generator code for `_SubjectCON_FUN_MP.gen.m` updates the properties of the `Subject` element. ← Code 5

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the subject.
%%%% idefault!
'SubjectCON_FUN_MP'

%%% iprop!
DESCRIPTION (constant, string) is the description of the subject.
%%%% idefault!
'Subject with connectivity and functional data (e.g. obtained from DTI and
     fMRI). The first layer contains a connectivity matrix and the second
     layer contains functional data.'

%%% iprop!
TEMPLATE (parameter, item) is the template of the subject.
%%% isettings!
'SubjectCON_FUN_MP'

%%% iprop!
ID (data, string) is a few-letter code for the subject.
%%%% idefault!
'SubjectCON_FUN_MP ID'

```

```
%%% iprop!
LABEL (metadata, string) is an extended label of the subject.
%%%% idefault!
'SubjectCON_FUN_MP label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the subject.
%%%% idefault!
'SubjectCON_FUN_MP notes'

%% iprops!

%%% iprop!
BA (data, item) is a brain atlas.
%%%% isettings!
'BrainAtlas'

%%% iprop!
CON (data, smatrix) is an adjacency matrix.
%%%% icheck_value!
br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
check = isequal(size(value), [br_number, br_number]); (1)
if check
    msg = 'All ok!';
else
    msg = ['CON must be a square matrix with the dimension equal to the
     number of brain regions (' int2str(br_number) ').'];
end
%%%% igui! (2)
pr = PanelPropMatrix('EL', sub, 'PROP', SubjectFUN.FUN, ...
    'ROWNAME', {'numbered'}, ...
    'COLUMNNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
    varargin{:});
```

(1) Same as in note (2) of Code 5.

(2) Same as in note (4) of Code 5.

Code 27: **SubjectCON_FUN_MP element tests.** The `tests` section from the element generator `_SubjectCON_FUN_MP.gen.m`. ← Code 6

```
%% itests!

%%% itest!
%%%% iname!
GUI
%%%% iprobability!
.01
%%%% icode!
im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx');
ba = im_ba.get('BA');

gr = Group('SUB_CLASS', 'SubjectCON_FUN_MP', 'SUB_DICT', IndexedDictionary('
     IT_CLASS', 'SubjectCON_FUN_MP'));
for i = 1:1:50 (1)
    sub = SubjectCON_FUN_MP( ...
        'ID', ['SUB CON ' int2str(i)], ...
        'LABEL', ['Subejct CON ' int2str(i)], ...
        'NOTES', ['Notes on subject CON ' int2str(i)], ...
        'BA', ba, ...
        'CON', rand(ba.get('BR_DICT').get('LENGTH')), ... (2)
        'FUN', rand(10, ba.get('BR_DICT').get('LENGTH')) ... (3)
        );
    sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
     rand()))
    sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
     CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
    gr.get('SUB_DICT').get('ADD', sub)
end

gui = GUIElement('PE', gr, 'CLOSEREQ', false);
gui.get('DRAW')
gui.get('SHOW')

gui.get('CLOSE')
```

(1) Same as in note (6) (7) of Code 3.

(2) constructs connectivity matrix.

(3) constructs functional matrix.

Implementation of a subject with structural data

Subject with structural data (SubjectST)

In this section we will show how to implement in detail `SubjectST`. The structural matrix can be obtained from sMRI data.

Code 28: **SubjectST element header.** The `header` section of the generator code for `_SubjectST.gen.m` provides the general information about the `SubjectST` element. ← Code 1

```
%% iheader!
SubjectST < Subject (sub, subject with structural data) is a subject with
     structural data (e.g. sMRI).

%%% idescription!
Subject with structural data (e.g. cortical thickness obtaibed from
     strcutural MRI) for each brain region.

%%% iseealso!
ImporterGroupSubjectST_TXT, ExporterGroupSubjectST_TXT,
     ImporterGroupSubjectST_XLS, ExporterGroupSubjectST_XLS
```

Code 29: **SubjectST element prop update.** The `props_update` section of the generator code for `_SubjectST.gen.m` updates the properties of the `Subject` element. ← Code 2

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the subject.
%%%% idefault!
'SubjectST'

%%% iprop!
DESCRIPTION (constant, string) is the description of the subject.
%%%% idefault!
'SubjectST with structural data (e.g. cortical thickness obtaibed from
     strcutural MRI) for each brain region.'

%%% iprop!
TEMPLATE (parameter, item) is the template of the subject.
%%% isettings!
'SubjectST'

%%% iprop!
ID (data, string) is a few-letter code for the subject.
%%%% idefault!
'SubjectST ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the subject.
%%%% idefault!
'SubjectST label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the subject.
```

```
%%%% idefault!
'SubjectST notes'

%% iprops!

%%% iprop!
BA (data, item) is a brain atlas.
%%%% isettings!
'BrainAtlas'

%%% iprop!
ST (data, cvector) is a column vector with data for each brain region.
%%%% icheck_value!
br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
check = (iscolumn(value) && isequal(size(value), [br_number, 1])) || (
     isempty(value) && br_number == 0); (1)
if check
    msg = 'All ok!';
else
    msg = ['ST must be a column vector with the same number of element as
     the brain regions (' int2str(br_number) ').'];
end
%%%% igui! (2)
pr = PanelPropMatrix('EL', sub, 'PROP', SubjectST.ST, ...
    'ROWNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
    'COLUMNNAME', {}, ...
    varargin{:});
```

(1) checks the size of the row of `value` is equal to the number of brain regions. The number of column is 1.

(2) Same as in note (4) of Code 2.

Code 30: **SubjectST element tests.** The `tests` section from the element generator `_SubjectST.gen.m`. ← Code 3

```
%% itests!

%%% itest!
%%%% iname!
GUI
%%%% iprobability!
.01
%%%% icode!
im_ba = ImporterBrainAtlasXLS('FILE', 'destrieux_atlas.xlsx');
ba = im_ba.get('BA');

gr = Group('SUB_CLASS', 'SubjectST', 'SUB_DICT', IndexedDictionary('IT_CLASS
     ', 'SubjectST'));
for i = 1:1:50
    sub = SubjectST( ...
        'ID', ['SUB ST ' int2str(i)], ...
        'LABEL', ['Subejct ST ' int2str(i)], ...
        'NOTES', ['Notes on subject ST ' int2str(i)], ...
        'BA', ba, ...
        'ST', rand(ba.get('BR_DICT').get('LENGTH'), 1) ... (1)
        );
    sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
     rand()))
    sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
     CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
    gr.get('SUB_DICT').get('ADD', sub)
end

gui = GUIElement('PE', gr, 'CLOSEREQ', false);
gui.get('DRAW')
gui.get('SHOW')

gui.get('CLOSE')
```

(1) constructs the random adjacency matrix with size of the number of brain regions by 1.

Subject with structural multiplex data (SubjectST_MP)

In this section we will show how to implement in detail `SubjectST_MP`. The structural matrix can be obtained from sMRI data.

Code 31: **SubjectST_MP element header.** The `header` section of the generator code for `_SubjectST_MP.gen.m` provides the general information about the `SubjectST_MP` element.← Code 4

```
%% iheader!
SubjectST_MP < Subject (sub, subject with structural multiplex data) is a
     subject with structural multiplex data (e.g. multiplex sMRI).

%%% idescription!
Subject with data for each brain region corresponding to L structural layers
      (e.g. cortical thickness obtained from structural MRI).

%%% iseealso!
ImporterGroupSubjectST_MP_TXT, ExporterGroupSubjectST_MP_TXT,
     ImporterGroupSubjectST_MP_XLS, ExporterGroupSubjectST_MP_XLS
```

Code 32: **SubjectST_MP element prop update.** The `props_update` section of the generator code for `_SubjectST_MP.gen.m` updates the properties of the `Subject` element. ← Code 5

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the subject.
%%%% idefault!
'SubjectST_MP'

%%% iprop!
DESCRIPTION (constant, string) is the description of the subject.
%%%% idefault!
'Subject with data for each brain region corresppponding to L structural
     layers (e.g. cortical thickness obtained from structural MRI).'

%%% iprop!
TEMPLATE (parameter, item) is the template of the subject.
%%% isettings!
'SubjectST_MP'

%%% iprop!
ID (data, string) is a few-letter code for the subject.
%%%% idefault!
'SubjectST_MP ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the subject.
%%%% idefault!
'SubjectST_MP label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the subject.
%%%% idefault!
'SubjectST_MP notes'

```

```
%% iprops!

%%% iprop!
BA (data, item) is a brain atlas.
%%%% isettings!
'BrainAtlas'

%%% iprop!
L (data, scalar) is the number of layers of subject data. (1)
%%%% idefault!
2

%%% iprop!
LAYERLABELS (metadata, stringlist) are the layer labels provided by the user
    . (2)

%%% iprop!
ALAYERLABELS (query, stringlist) returns the processed layer labels. (3)
%%%% icalculate!
value = sub.get('LAYERLABELS');

%%% iprop!
ST_MP (data, cell) is a cell containing L vectors, each with data for each
    brain region.
%%%% icheck_value!
br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
num_layers = sub.get('L');
check = (iscell(value) && isequal(length(value), num_layers)  && isequal(
    cellfun(@(v) size(v, 1), value), ones(1, num_layers) * br_number)) || (
    isempty(value) && br_number == 0);  (4)
if check
    msg = 'All ok!';
else
    msg = ['ST_MP must be a column vector with the same number of element as
      the brain regions (' int2str(br_number) ').'];
end
%%%% igui! (5)
pr = PanelPropCell('EL', sub, 'PROP', SubjectST_MP.ST_MP, ...
    'TABLE_HEIGHT', s(40), ...
    'XSLIDERSHOW', true, ...
    'XSLIDERLABELS', sub.getCallback('ALAYERLABELS'), ...
    'YSLIDERSHOW', false, ...
    'ROWNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
    'COLUMNNAME', {}, ...
    varargin{:});
```

(1) Same as in note (1) of Code 5.

(2) Same as in note (2) of Code 5.

(3) Same as in note (3) of Code 5.

(4) checks the size of each layer are equal to the number of brain regions. The size of each layer is the number of regions by 1.

(5) Same as in note (8) (9) (10) (11) of Code 5.

Code 33: **SubjectST_MP element tests.** The `tests` section from the element generator `_SubjectST_MP.gen.m`. ← Code 6

```

%% itests!

%%% itest!
%%%% iname!
GUI
%%%% iprobability!
.01
%%%% icode!
im_ba = ImporterBrainAtlasXLS('FILE', 'destrieux_atlas.xlsx');
```

```
ba = im_ba.get('BA');

gr = Group('SUB_CLASS', 'SubjectST_MP', 'SUB_DICT', IndexedDictionary('
     IT_CLASS', 'SubjectST_MP'));
for i = 1:1:10 (1)
    sub = SubjectST_MP( ...
        'ID', ['SUB ST_MP ' int2str(i)], ...
        'LABEL', ['Subejct ST_MP ' int2str(i)], ...
        'NOTES', ['Notes on subject ST_MP ' int2str(i)], ...
        'BA', ba, ...
        'L', 3, ...
        'LAYERLABELS', {'L1' 'L2' 'L3'}, ...
        'ST_MP', {rand(ba.get('BR_DICT').get('LENGTH'), 1), rand(ba.get('
     BR_DICT').get('LENGTH'), 1), rand(ba.get('BR_DICT').get('LENGTH'), 1)}
     ...
        );
    sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
     rand()))
    sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
     CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
    gr.get('SUB_DICT').get('ADD', sub)
end

gui = GUIElement('PE', gr, 'CLOSEREQ', false);
gui.get('DRAW')
gui.get('SHOW')

gui.get('CLOSE')
```

(1) Same as in note (1) (2) (3) of Code 6.