

Implement, export and import groups of subjects

The BRAPH 2 Developers

September 21, 2023

This is the developer tutorial for implementing, exporting and importing groups of subjects. We will explain how to create generator files *.gen.m for new subjects and well as to export and import them. All *.gen.m files can then be compiled by braph2genesis. All types of subjects are extensions of the base element Subject. 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 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

<i>Implementation of a subject with connectivity data</i>	<i>2</i>
<i>Subject with connectivity data (SubjectCON)</i>	<i>2</i>
<i>Subject with connectivity multiplex data (SubjectCON_MP)</i>	<i>5</i>
<i>Implementation of Importer and Exporter of the data (for SubjectCON)</i>	<i>8</i>
<i>Importer of a CON subject group from TXT (ImporterGroupSubjectCON_TXT)</i>	<i>8</i>
<i>Importer of a CON subject group from XLS/XLSX (ImporterGroupSubjectCON_XLS)</i>	<i>16</i>
<i>Exporter of a CON subject group from TXT (ExporterGroupSubjectCON_TXT)</i>	<i>23</i>
<i>Exporter of a CON subject group from XLS/XLSX (ExporterGroupSubjectCON_XLS)</i>	<i>31</i>
<i>Implementation of a subject with functional data</i>	<i>39</i>
<i>Subject with functional data (SubjectFUN)</i>	<i>39</i>
<i>Subject with functional multiplex data (SubjectFUN_MP)</i>	<i>42</i>
<i>Implementation of a subject with connectivity and functional data</i>	<i>45</i>
<i>Subject with connectivity and functional multiplex data (SubjectCON_FUN_MP)</i>	<i>45</i>
<i>Implementation of a subject with structural data</i>	<i>48</i>
<i>Subject with structural data (SubjectST)</i>	<i>48</i>
<i>Subject with structural multiplex data (SubjectST_MP)</i>	<i>51</i>

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.

```

1 %% iheader!
2 SubjectCON < Subject (sub, subject with connectivity matrix) is a subject
   with connectivity matrix (e.g. DTI). ①
3
4 %%% idescription!
5 Subject with a connectivity matrix (e.g. obtained from DTI).
6
7 %%% iseealso! ②
8 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.

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the subject.
5 %%% idefault!
6 'SubjectCON'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the subject.
10 %%% idefault!
11 'SubjectCON with a connectivity matrix (e.g. obtained from DTI).'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the subject.
15 %%% isettings!
16 'SubjectCON'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the subject.
20 %%% idefault!
21 'SubjectCON ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the subject.
25 %%% idefault!
26 'SubjectCON label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the subject.
30 %%% idefault!

```

```

31 'SubjectCON notes'
32
33 %% iprops!
34
35 %% iprop!
36 BA (data, item) is a brain atlas.
37 %%%% isettings!
38 'BrainAtlas'
39
40
41 %% iprop!
42 CON (data, smatrix) is an adjacency matrix.
43 %%%% icheck_value!
44 br_number = sub.get('BA').get('BR_DICT').get('LENGTH'); ①
45 check = isequal(size(value), [br_number, br_number]); ②
46 if check ③
47     msg = 'All ok!';
48 else
49     msg = ['CON must be a square matrix with the dimension equal to the
50           number of brain regions (' int2str(br_number) ').'];
51 end
52 %%%% igui! ④
53 pr = PanelPropMatrix('EL', sub, 'PROP', SubjectCON.CON, ...
54     'ROWNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
55     'COLUMNNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
56     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.

```

1 %% itests!
2
3 %%% itest!
4 %%% iname!
5 GUI ①
6 %%% iprobability! ②
7 .01
8 %%% icode!
9 im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx'); ③
10 ba = im_ba.get('BA'); ④
11
12 gr = Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('
    IT_CLASS', 'SubjectCON')); ⑤
13 for i = 1:1:50 ⑥
14     sub = SubjectCON( ... ⑦
15         'ID', ['SUB CON ' int2str(i)], ...
16         'LABEL', ['Subejct CON ' int2str(i)], ...
17         'NOTES', ['Notes on subject CON ' int2str(i)], ...
18         'BA', ba, ...
19         'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
20     );
21     sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
        rand())) ⑧
22     sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
        CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1))) ⑨
23     gr.get('SUB_DICT').get('ADD', sub) ⑩
24 end
25
26 gui = GUIElement('PE', gr, 'CLOSEREQ', false); ⑪
27 gui.get('DRAW') ⑫
28 gui.get('SHOW') ⑬
29
30 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 multi-layer 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](#)

```

1
2 %% iheader!
3 SubjectCON_MP < Subject (sub, subject with connectivity multiplex data) is a
  subject with connectivity multiplex data.
4
5 %%% idescription!
6 Subject with L connectivity matrices (e.g. obtained from DTI).
7
8 %%% iseealso!
9 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](#)

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the subject.
5 %%% idefault!
6 'SubjectCON_MP'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the subject.
10 %%% idefault!
11 'Subject with L connectivity matrices (e.g. obtained from DTI).'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the subject.
15 %%% isettings!
16 'SubjectCON_MP'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the subject.
20 %%% idefault!
21 'SubjectCON_MP ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the subject.
25 %%% idefault!
26 'SubjectCON_MP label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the subject.
30 %%% idefault!

```

```

31 'SubjectCON_MP notes'
32
33 %% iprops!
34
35 %%% iprop!
36 BA (data, item) is a brain atlas.
37 %%% isettings!
38 'BrainAtlas'
39
40 %%% iprop!
41 L (data, scalar) is the number of layers of subject data. ①
42 %%% ndefault!
43 2 ②
44
45 %%% iprop!
46 LAYERLABELS (metadata, stringlist) are the layer labels provided by the user
   . ③
47
48 %%% iprop!
49 ALAYERLABELS (query, stringlist) returns the processed layer labels. ④
50 %%% icalculate!
51 value = sub.get('LAYERLABELS'); ⑤
52
53 %%% iprop!
54 CON_MP (data, cell) is a cell containing L matrices corresponding
   connectivity matrices of each layer.
55 %%% ickheck_value!
56 br_number = sub.get('BA').get('BR-DICT').get('LENGTH');
57 num_layers = sub.get('L'); ⑥
58 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
59 if check
60     msg = 'All ok!';
61 else
62     msg = ['CON_MP must be a cell with L square matrices with the dimension
   equal to the number of brain regions (' int2str(br_number) ').'];
63 end
64 %%% igui!
65 pr = PanelPropCell('EL', sub, 'PROP', SubjectCON_MP.CON_MP, ...
66     'TABLE_HEIGHT', s(40), ... ⑧
67     'XSLIDERSHOW', true, ... ⑨
68     'XSLIDERLABELS', sub.getCallback('ALAYERLABELS'), ... ⑩
69     'YSLIDERSHOW', false, ... ⑪
70     'ROWNAME', sub.get('BA').get('BR-DICT').getCallback('KEYS'), ...
71     'COLUMNNAME', sub.get('BA').get('BR-DICT').getCallback('KEYS'), ...
72     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](#)

```

1 %% itests!
2
3 %%% itest!
4 %%% iname!
5 GUI

```

```

6 %%% iprobability!
7 .01
8 %%% icode!
9 im_ba = ImporterBrainAtlasXLS('FILE', 'aal90_atlas.xlsx');
10 ba = im_ba.get('BA');
11
12 gr = Group('SUB_CLASS', 'SubjectCON_MP', 'SUB_DICT', IndexedDictionary('
    IT_CLASS', 'SubjectCON_MP'));
13 for i = 1:1:10
14     sub = SubjectCON_MP( ...
15         'ID', ['SUB CON_MP ' int2str(i)], ...
16         'LABEL', ['Subejct CON_MP ' int2str(i)], ...
17         'NOTES', ['Notes on subject CON_MP ' int2str(i)], ...
18         'BA', ba, ...
19         'L', 3, ... ①
20         'LAYERLABELS', {'L1' 'L2' 'L3'}, ... ②
21         'CON_MP', {rand(ba.get('BR_DICT').get('LENGTH')), rand(ba.get('
BR_DICT').get('LENGTH')), rand(ba.get('BR_DICT').get('LENGTH'))} ...
22     ); ③
23     sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
        rand()))
24     sub.memorize('VOI_DICT').get('ADD', VOICategorical('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 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 %% iheader!
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 %%% idescription!
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 existng)
   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, SunjectCON, 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 %% iprops_update!
3
4 %%% iprop!
5 NAME (constant, string) is the name of the CON subject group importer from
   TXT.
6 %%% idefault!
7 'ImporterGroupSubjectCON_TXT'
8
9 %%% iprop!
10 DESCRIPTION (constant, string) is the description of the CON subject group
   importer from TXT.
11 %%% idefault!
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_brph2'))
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 %%% icalculate!
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 %%% icalculate! (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
95 ), 'Delimiter', '\t'), 1); (15)
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
100                 (16)
101                 end
102                 end
103
104                 sub_dict = gr.memorize('SUB_DICT'); (17)
105                 for i = 1:length(files)
106                     braph2waitbar(wb, .15 + .85 * i / length(files), ['Loading
107 subject ' num2str(i) ' of ' num2str(length(files)) ' ...']) (18)
108
109                     [~, sub_id] = fileparts(files(i).name);
110                     CON = table2array(readtable(fullfile(directory, files(i).
111 name), 'Delimiter', '\t')); (19)
112                     if size(CON, 1) ~= ba.get('BR_DICT').get('LENGTH') || size(
113 CON, 2) ~= ba.get('BR_DICT').get('LENGTH') (20)
114                         error( ...
115                             [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO], ...
116                             [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO '\n']
117 ...
118                             '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.

```

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

```

1
2 %% itests!
3
4 %% iexcluded_props! ①
5 [ImporterGroupSubjectCON_TXT.GET_DIR]
6
7 %% itest!
8 %%% iname!
9 Create example files ②
10 %%% icode!
11 data_dir = [fileparts(which('SubjectCON')) filesep 'Example data CON TXT'];
12   ③
13 if ~isdir(data_dir)
14     mkdir(data_dir); ④
15
16 % Brain Atlas
17 im_ba = ImporterBrainAtlasTXT('FILE', 'desikan_atlas.txt'); ⑤
18 ba = im_ba.get('BA');
19 ex_ba = ExporterBrainAtlasTXT( ... ⑥
20     'BA', ba, ...
21     'FILE', [data_dir filesep() 'atlas.txt'] ...
22 );
23 ex_ba.get('SAVE')
24 N = ba.get('BR_DICT').get('LENGTH'); ⑦
25
26 % saves RNG
27 rng_settings_ = rng(); rng('default') ⑧
28 \code{rng}
29 sex_options = {'Female' 'Male'};
30
31 % Group 1 ⑨
32 K1 = 2; ⑩
33 beta1 = 0.3; ⑪
34 gr1_name = 'CON_Group_1_TXT';
35 gr1_dir = [data_dir filesep() gr1_name];
36 mkdir(gr1_dir);
37 vois1 = [ ⑫
38     {'Subject ID'} {'Age'} {'Sex'}}
39     {{}} {{}} {{{' sprintf(' '%s' ', sex_options{:}) '}}}}
40 ];
41 for i = 1:1:50 % subject number ⑬
42     sub_id = ['SubjectCON_MP_' num2str(i)];
43
44     h1 = WattsStrogatz(N, K1, beta1); ⑭
45
46     A1 = full(adjacency(h1)); A1(1:length(A1)+1:numel(A1)) = 0; ⑮
47     r = 0 + (0.5 - 0)*rand(size(A1)); diffA = A1 - r; A1(A1 ~= 0) =
48     diffA(A1 ~= 0); ⑯
49     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     writetable(array2table(A1), [gr1_dir filesep() sub_id '.txt'], '
49     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'], '
54     Delimiter', '\t', 'WriteVariableNames', false) (20)
55
56 % Group 2 (21)
57 K2 = 2;
58 beta2 = 0.85;
59 gr2_name = 'CON_Group_2_TXT';
60 gr2_dir = [data_dir filesep() gr2_name];
61 mkdir(gr2_dir);
62 vois2 = [
63     {{'Subject ID'} {'Age'} {'Sex'}}
64     {{}} {{['{' sprintf(' %s' , sex_options{:}) '']}}}
65 ];
66 for i = 51:1:100
67     sub_id = ['SubjectCON_MP_' num2str(i)];
68
69     h2 = WattsStrogatz(N, K2, beta2);
70
71     A2 = full(adjacency(h2)); A2(1:length(A2)+1:numel(A2)) = 0;
72     r = 0 + (0.5 - 0)*rand(size(A2)); diffA = A2 - r; A2(A2 ~= 0) =
73     diffA(A2 ~= 0);
74     A2 = max(A2, transpose(A2));
75
76     writetable(array2table(A2), [gr2_dir filesep() 'SubjectCON_' num2str
77     (i) '.txt'], 'Delimiter', '\t', 'WriteVariableNames', false)
78
79     % variables of interest
80     vois2 = [vois2; {sub_id, randi(90), sex_options(randi(2))}];
81 end
82 writetable(table(vois2), [data_dir filesep() gr2_name '.vois.txt'], '
83     Delimiter', '\t', 'WriteVariableNames', false)
84
85 % reset RNG
86 rng(rng_settings_) (22)
87 end
88
89 %%% itest_functions!
90 function h = WattsStrogatz(N,K,beta) (23)
91 % H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
92 % nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
93 %
94 % beta = 0 is a ring lattice, and beta = 1 is a random graph.
95
96 % Connect each node to its K next and previous neighbors. This constructs
97 % indices for a ring lattice.
98 s = repelem((1:N)',1,K); (24)
99 t = s + repmat(1:K,N,1); (25)
100 t = mod(t-1,N)+1; (26)
101
102 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](#)

```

1
2 %% iheader!
3 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.
4
5 %%% idescription!
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 existng)
    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, 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](#)

```

1
2 %% iprops_update!
3
4 %%% iprop!
5 NAME (constant, string) is the name of the CON subject group importer from
    XLS/XLSX.
6 %%% idefault!
7 'ImporterGroupSubjectCON_XLS'
8
9 %%% iprop!
10 DESCRIPTION (constant, string) is the description of the CON subject group
    importer from XLS/XLSX.
11 %%% idefault!
12 '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.'
13
14 %%% iprop!
15 TEMPLATE (parameter, item) is the template of the CON subject group importer
    from XLS/XLSX.

```

```

16 %%%% isettings!
17 'ImporterGroupSubjectCON_XLS'
18
19 %%% iprop!
20 ID (data, string) is a few-letter code for the CON subject group importer
    from XLS/XLSX.
21 %%%% idefault!
22 'ImporterGroupSubjectCON_XLS ID'
23
24 %%% iprop!
25 LABEL (metadata, string) is an extended label of the CON subject group
    importer from XLS/XLSX.
26 %%%% idefault!
27 'ImporterGroupSubjectCON_XLS label'
28
29 %%% iprop!
30 NOTES (metadata, string) are some specific notes about the CON subject group
    importer from XLS/XLSX.
31 %%%% idefault!
32 'ImporterGroupSubjectCON_XLS 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_brph2'))
40
41 %%% iprop!
42 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.
43 %%%% isettings!
44 'ImporterGroupSubjectCON_XLS'
45 %%%% icalculate!
46 directory = uigetdir('Select directory');
47 if ischar(directory) && isfolder(directory)
48     im.set('DIRECTORY', directory);
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 %%%% icalculate!
62 check = any(strcmp(value.get(Group.SUB_CLASS_TAG), subclasses('SubjectCON',
    [], [], true))); ①
63 %%%% idefault!
64 Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', '
    SubjectCON')) ②
65 %%%% icalculate! ③
66 gr = Group( ...
67     'SUB_CLASS', 'SubjectCON', ...
68     '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,
89             '*.xls'))];
90
91         if ~isempty(files)
92             % brain atlas
93             ba = im.get('BA');
94             if ba.get('BR-DICT').get('LENGTH') == 0
95                 % adds the number of regions of the first file to the brain
96                 atlas
97                     br_number = size(xlsread(fullfile(directory, files(1).name))
98                         , 1);
99                     br_dict = ba.memorize('BR-DICT');
100                     for j = 1:1:br_number
101                         br_dict.get('ADD', BrainRegion('ID', ['br' int2str(j)]))
102                     end
103
104                 % adds subjects
105                 sub_dict = gr.memorize('SUB-DICT');
106                 for i = 1:1:length(files)
107                     braph2waitbar(wb, .15 + .85 * i / length(files), ['Loading
108                         subject ' num2str(i) ' of ' num2str(length(files)) ' ...'])
109
110                     % read file
111                     [~, sub_id] = fileparts(files(i).name);
112
113                     CON = xlsread(fullfile(directory, files(i).name));
114                     if size(CON, 1) ~= ba.get('BR-DICT').get('LENGTH') || size(
115                         CON, 2) ~= ba.get('BR-DICT').get('LENGTH')
116                         error( ...
117                             [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO], ...
118                             [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO '\n']
119                             ...
120                             'The file ' sub_id ' should contain a matrix '
121                             int2str(ba.get('BR-DICT').get('LENGTH')) 'x' int2str(ba.get('BR-DICT').
122                                 get('LENGTH')) ', ' ...
123                             'while it is ' int2str(size(CON, 1)) 'x' int2str(
124                                 size(CON, 2)) '...'
125                             )
126                     end
127
128                     sub = SubjectCON( ...

```

```

121         'ID', sub_id, ...
122         'BA', ba, ...
123         'CON', CON ...
124     );
125     sub_dict.get('ADD', sub);
126 end
127
128 % variables of interest
129 vois = [];
130 if isfile([directory '.vois.xls'])
131     [~, ~, vois] = xlsread([directory '.vois.xls']);
132 elseif isfile([directory '.vois.xlsx'])
133     [~, ~, vois] = xlsread([directory '.vois.xlsx']);
134 end
135 if ~isempty(vois)
136     for i = 3:1:size(vois, 1)
137         sub_id = vois{i, 1};
138         sub = sub_dict.get('IT', sub_id);
139         for v = 2:1:size(vois, 2)
140             voi_id = vois{1, v};
141             if isnumeric(vois{2, v}) % VOINumeric
142                 sub.memorize('VOI_DICT').get('ADD', ...
143                     VOINumeric( ...
144                         'ID', voi_id, ...
145                         'V', vois{i, v} ...
146                     ) ...
147                 );
148             elseif ischar(vois{2, v}) % VOICategorical
149                 sub.memorize('VOI_DICT').get('ADD', ...
150                     VOICategorical( ...
151                         'ID', voi_id, ...
152                         'CATEGORIES', str2cell(vois{2, v}), ...
153                         'V', find(strcmp(vois{i, v}, str2cell(
154                             vois{2, v}))) ...
155                     ) ...
156                 );
157             end
158         end
159     end
160 end
161 catch e
162     braph2waitbar(wb, 'close')
163     rethrow(e)
164 end
165
166 braph2waitbar(wb, 'close')
167 else
168     error([BRAPH2.STR ':ImporterGroupSubjectCON_XLS:' BRAPH2.ERR_IO], ...
169         [BRAPH2.STR ':ImporterGroupSubjectCON_XLS:' BRAPH2.ERR_IO '\n' ...
170         'The prop DIRECTORY must be an existing directory, but it is ''
171         directory ''''] ...
172     );
173 end
174
175 value = gr;

```

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

```

1
2 %% itests!
3
4 %%% iexcluded_props!
5 [ImporterGroupSubjectCON_XLS.GET_DIR]
6
7 %%% itest!
8 %%% iname!
9 Create example files
10 %%% icode!
11 data_dir = [fileparts(which('SubjectCON')) filesep('Example data CON XLS')];
12 if ~isdir(data_dir)
13     mkdir(data_dir);
14
15 % Brain Atlas
16 im_ba = ImporterBrainAtlasXLS('FILE', 'desikan-atlas.xlsx');
17 ba = im_ba.get('BA');
18 ex_ba = ExporterBrainAtlasXLS( ...
19     'BA', ba, ...
20     'FILE', [data_dir filesep() 'atlas.xlsx'] ...
21 );
22 ex_ba.get('SAVE')
23 N = ba.get('BR_DICT').get('LENGTH');
24
25 % saves RNG
26 rng_settings_ = rng(); rng('default')
27
28 sex_options = {'Female' 'Male'};
29
30 % Group 1
31 K1 = 2;
32 betal = 0.3;
33 gr1_name = 'CON_Group_1_XLS';
34 gr1_dir = [data_dir filesep() gr1_name];
35 mkdir(gr1_dir);
36 vois1 = [
37     {'Subject ID'} {'Age'} {'Sex'}}
38     {{}} {} cell2str(sex_options)}
39 ];
40 for i = 1:1:50 % subject number
41     sub_id = ['SubjectCON_' num2str(i)];
42
43     h1 = WattsStrogatz(N, K1, betal); % create two WS graph
44
45     A1 = full(adjacency(h1)); A1(1:length(A1)+1:numel(A1)) = 0;
46     r = 0 + (0.5 - 0)*rand(size(A1)); diffA = A1 - r; A1(A1 ~= 0) =
47     diffA(A1 ~= 0);
48     A1 = max(A1, transpose(A1)); % make the adjacency matrix symmetric
49
50     writetable(array2table(A1), [gr1_dir filesep() sub_id '.xlsx'], '
51     WriteVariableNames', false)
52     vois1 = [vois1; {sub_id, randi(90), sex_options(randi(2))}];
53 end
54 writetable(table(vois1), [data_dir filesep() gr1_name '.vois.xlsx'], '
55     WriteVariableNames', false)
56
57 % Group 2

```

```

55     K2 = 2;
56     beta2 = 0.85;
57     gr2_name = 'CON_Group_2_XLS';
58     gr2_dir = [data_dir filesep() gr2_name];
59     mkdir(gr2_dir);
60     vois2 = [
61         {'Subject ID'} {'Age'} {'Sex'}
62         {} {} cell2str(sex_options)}
63     ];
64     for i = 51:1:100
65         sub_id = ['SubjectCON_' num2str(i)];
66
67         h2 = WattsStrogatz(N, K2, beta2);
68         % figure(2)
69         % plot(h2, 'NodeColor',[1 0 0], 'EdgeColor',[0 0 0], 'EdgeAlpha
        % ,0.1, 'Layout','circle');
70         % title(['Group 2: Graph with $N = $ ' num2str(N_nodes) ...
71         % ' nodes, $K = $ ' num2str(K2) ', and $\beta = $ ' num2str(
        beta2)], ...
72         % 'Interpreter','latex')
73         % axis equal
74
75         A2 = full(adjacency(h2)); A2(1:length(A2)+1:numel(A2)) = 0;
76         r = 0 + (0.5 - 0)*rand(size(A2)); diffA = A2 - r; A2(A2 ~= 0) =
        diffA(A2 ~= 0);
77         A2 = max(A2, transpose(A2));
78
79         writetable(array2table(A2), [gr2_dir filesep() sub_id '.xlsx'], '
        WriteVariableNames', false)
80
81         % variables of interest
82         vois2 = [vois2; {sub_id, randi(90), sex_options(randi(2))}];
83     end
84     writetable(table(vois2), [data_dir filesep() gr2_name '.vois.xlsx'], '
        WriteVariableNames', false)
85
86     % reset RNG
87     rng(rng_settings_)
88 end
89
90 %%% itest_functions!
91 function h = WattsStrogatz(N,K,beta)
92 % H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
93 % nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
94 %
95 % beta = 0 is a ring lattice, and beta = 1 is a random graph.
96
97 % Connect each node to its K next and previous neighbors. This constructs
98 % indices for a ring lattice.
99 s = repelem((1:N)',1,K);
100 t = s + repmat(1:K,N,1);
101 t = mod(t-1,N)+1;
102
103 % Rewire the target node of each edge with probability beta
104 for source=1:N
105     switchEdge = rand(K, 1) < beta;
106
107     newTargets = rand(N, 1);
108     newTargets(source) = 0;
109     newTargets(s(t==source)) = 0;
110     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!

```

```

17 'ExporterGroupSubjectCON.TXT'
18
19 %% iprop!
20 ID (data, string) is a few-letter code for the CON subject group exporter in
    TXT.
21 %%%% idefault!
22 'ExporterGroupSubjectCON.TXT ID'
23
24 %% iprop!
25 LABEL (metadata, string) is an extended label of the CON subject group
    exporter in TXT.
26 %%%% idefault!
27 'ExporterGroupSubjectCON.TXT label'
28
29 %% iprop!
30 NOTES (metadata, string) are some specific notes about the CON subject group
    exporter in TXT.
31 %%%% idefault!
32 'ExporterGroupSubjectCON.TXT notes'
33
34 %% iprops!
35
36 %% iprop!
37 GR (data, item) is a group of subjects with connectivity data.
38 %%%% isettings!
39 'Group'
40 %%%% icalculate!
41 check = any(strcmp(value.get(Group.SUB_CLASS_TAG), subclasses('SubjectCON',
    [], [], true))); ①
42 %%%% idefault!
43 Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', '
    SubjectCON'))
44
45 %% iprop!
46 DIRECTORY (data, string) is the directory name where to save the group of
    subjects with connectivity data.
47 %%%% idefault!
48 [fileparts(which('test_brph2')) filesep '
    default_group_subjects_CON_most_likely_to_be_erased'] ②
49
50 %% iprop!
51 PUT_DIR (query, item) opens a dialog box to set the directory where to save
    the group of subjects with connectivity data.
52 %%%% isettings!
53 'ExporterGroupSubjectCON.TXT'
54 %%%% icalculate!
55 directory = uigetdir('Select directory'); ③
56 if ischar(directory) && isfolder(directory) ④
57     ex.set('DIRECTORY', directory);
58 end
59 value = ex;
60
61 %% iprop!
62 SAVE (result, empty) saves the group of subjects with connectivity data in
    TXT files in the selected directory.
63 %%%% icalculate!
64 directory = ex.get('DIRECTORY');
65
66 if isfolder(directory) ⑤
67     wb = brph2waitbar(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 '
83         num2str(i) ' of ' num2str(sub_number) '...']) (7)
84
85     sub = sub_dict.get('IT', i); (8)
86     sub_id = sub.get('ID'); (9)
87     sub_CON = sub.get('CON'); (10)
88
89     tab = table(sub_CON); (11)
90
91     sub_file = [gr_directory filesep() sub_id '.txt'];
92
93     % save file
94     writetable(tab, sub_file, 'Delimiter', '\t', 'WriteVariableNames',
95         false); (12)
96 end
97
98 % variables of interest
99 voi_ids = {};
100 for i = 1:1:sub_number
101     sub = sub_dict.get('IT', i);
102     voi_ids = unique([voi_ids, sub.get('VOI_DICT').get('KEYS')]); (13)
103 end
104 if ~isempty(voi_ids)
105     vois = cell(2 + sub_number, 1 + length(voi_ids));
106     vois{1, 1} = 'Subject ID';
107     vois(1, 2:end) = voi_ids;
108     for i = 1:1:sub_number
109         sub = sub_dict.get('IT', i);
110         vois{2 + i, 1} = sub.get('ID');
111
112         voi_dict = sub.get('VOI_DICT');
113         for v = 1:1:voi_dict.get('LENGTH') (14)
114             voi = voi_dict.get('IT', v);
115             voi_id = voi.get('ID');
116             if isa(voi, 'VOINumeric') % Numeric
117                 vois{2 + i, 1 + find(strcmp(voi_id, voi_ids))} = voi.get
118                     ('V');
119             elseif isa(voi, 'VOICategorical') % Categorical
120                 categories = voi.get('CATEGORIES');
121                 vois{2, 1 + find(strcmp(voi_id, voi_ids))} = {'{'
122                     sprintf(' '%s' ', categories{:}) '}'}};
123                 vois{2 + i, 1 + find(strcmp(voi_id, voi_ids))} =
124                     categories{voi.get('V')};
125             end
126         end
127     end
128 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`.

```

1
2 %% itests!
3
4 %%% iexcluded_props! ①
5 [ExporterGroupSubjectCON_TXT.PUT_DIR]
6
7 %%% itest!
8 %%% iname!
9 Delete directory TBE ②
10 %%% iprobability!
11 1
12 %%% icode!
13 warning('off', 'MATLAB:DELETE:FileNotFound')
14 dir_to_be_erased = ExporterGroupSubjectCON_TXT.getPropDefault('DIRECTORY');
15 if isfolder(dir_to_be_erased)
16     rmdir(dir_to_be_erased, 's')
17 end
18 warning('on', 'MATLAB:DELETE:FileNotFound')
19
20 %%% itest!
21 %%% iname!
22 Export and import ③
23 %%% iprobability!
24 .01
25 %%% icode!
26 br1 = BrainRegion( ... ④
27     'ID', 'ISF', ...
28     'LABEL', 'superiorfrontal', ...
29     'NOTES', 'notes1', ...
30     'X', -12.6, ...
31     'Y', 22.9, ...
32     'Z', 42.4 ...
33 );
34 br2 = BrainRegion( ...
35     'ID', 'LFP', ...
36     'LABEL', 'frontalpole', ...
37     'NOTES', 'notes2', ...
38     'X', -8.6, ...
39     'Y', 61.7, ...
40     'Z', -8.7 ...
41 );
42 br3 = BrainRegion( ...
43     'ID', 'LRMF', ...
44     'LABEL', 'rostralmiddlefrontal', ...
45     'NOTES', 'notes3', ...
46     'X', -31.3, ...
47     'Y', 41.2, ...
48     'Z', 16.5 ...
49 );
50 br4 = BrainRegion( ...
51     'ID', 'LCMF', ...
52     'LABEL', 'caudalmiddlefrontal', ...
53     'NOTES', 'notes4', ...
54     'X', -34.6, ...
55     'Y', 10.2, ...
56     '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'.

```

57 );
58 br5 = BrainRegion( ...
59   'ID', 'LPOB', ...
60   'LABEL', 'parsorbitalis', ...
61   'NOTES', 'notes5', ...
62   'X', -41, ...
63   'Y', 38.8, ...
64   'Z', -11.1 ...
65 );
66
67 ba = BrainAtlas( ...
68   'ID', 'TestToSaveCoolID', ...
69   'LABEL', 'Brain Atlas', ...
70   'NOTES', 'Brain atlas notes', ...
71   'BR_DICT', IndexedDictionary('IT_CLASS', 'BrainRegion', 'IT_LIST', {br1,
72     br2, br3, br4, br5}) ... ⑤
73 );
74
75 sub1 = SubjectCON( ... ⑥
76   'ID', 'SUB CON 1', ...
77   'LABEL', 'Subejct CON 1', ...
78   'NOTES', 'Notes on subject CON 1', ...
79   'BA', ba, ...
80   'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
81 );
82 sub1.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 75)) ⑦
83 sub1.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES',
84   {'Female', 'Male'}, 'V', find(strcmp('Female', {'Female', 'Male'}))))
85   ⑧
86
87 sub2 = SubjectCON( ...
88   'ID', 'SUB CON 2', ...
89   'LABEL', 'Subejct CON 2', ...
90   'NOTES', 'Notes on subject CON 2', ...
91   'BA', ba, ...
92   'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
93 );
94 sub2.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 70))
95 sub2.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES',
96   {'Female', 'Male'}, 'V', find(strcmp('Male', {'Female', 'Male'}))))
97
98 sub3 = SubjectCON( ...
99   'ID', 'SUB CON 3', ...
100   'LABEL', 'Subejct CON 3', ...
101   'NOTES', 'Notes on subject CON 3', ...
102   'BA', ba, ...
103   'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
104 );
105 sub3.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 50))
106 sub3.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES',
107   {'Female', 'Male'}, 'V', find(strcmp('Female', {'Female', 'Male'}))))
108
109 gr = Group( ...
110   'ID', 'GR CON', ...
111   'LABEL', 'Group label', ...
112   'NOTES', 'Group notes', ...
113   'SUB_CLASS', 'SubjectCON', ...
114   'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON', 'IT_LIST', {sub1,
115     sub2, sub3}) ... ⑨
116 );

```

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_brph2')) 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)
      'DIRECTORY', directory, ...
      'GR', gr ...
      );
121 ex.get('SAVE');
122
123 im1 = ImporterGroupSubjectCON_TXT( ... (12)
      'DIRECTORY', [directory filesep() gr.get(Group.ID)], ...
      'BA', ba ...
      );
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.
139         get('VOI_DICT').get('IT', 'Age').get('V')) & ...
140         isequal(sub.get('VOI_DICT').get('IT', 'Sex').get('V'), sub_loaded.
141         get('VOI_DICT').get('IT', 'Sex').get('V')) & ...
142         isequal(round(sub.get('CON'), 10), round(sub_loaded.get('CON'), 10))
143         , ...
144         [BRAPH2.STR ':ExporterGroupSubjectCON_TXT:' BRAPH2.FAIL_TEST], ...
145         'Problems saving or loading a group.')
146 end
147
148 im2 = ImporterGroupSubjectCON_TXT( ... (16)
      'DIRECTORY', [directory filesep() gr.get(Group.ID)] ...
      );
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') ⑪

```

⑪ 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 %% iheader!
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 %%% idescription!
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 %%% iseealso!
9 Group, SunjectCON, 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 %% iprops_update!
3
4 %%% iprop!
5 NAME (constant, string) is the name of the CON subject group exporter in
  XLSX.
6 %%% idefault!
7 'ExporterGroupSubjectCON_XLS'
8
9 %%% iprop!
10 DESCRIPTION (constant, string) is the description of the CON subject group
  exporter in XLSX.
11 %%% idefault!
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 %%% iprop!
15 TEMPLATE (parameter, item) is the template of the CON subject group exporter
  in XLSX.

```

```

16 %%%% isettings!
17 'ExporterGroupSubjectCON_XLS'
18
19 %%% iprop!
20 ID (data, string) is a few-letter code for the CON subject group exporter in
    XLSX.
21 %%%% idefault!
22 'ExporterGroupSubjectCON_XLS ID'
23
24 %%% iprop!
25 LABEL (metadata, string) is an extended label of the CON subject group
    exporter in XLSX.
26 %%%% idefault!
27 'ExporterGroupSubjectCON_XLS label'
28
29 %%% iprop!
30 NOTES (metadata, string) are some specific notes about the CON subject group
    exporter in XLSX.
31 %%%% idefault!
32 'ExporterGroupSubjectCON_XLS notes'
33
34 %%% iprops!
35
36 %%% iprop!
37 GR (data, item) is a group of subjects with connectivity data.
38 %%%% isettings!
39 'Group'
40 %%%% icheck_value!
41 check = any(strcmp(value.get(Group.SUB_CLASS_TAG), subclasses('SubjectCON',
    [], [], true))); % Format.checkFormat(Format.ITEM, value, 'Group')
    already checked
42 %%%% idefault! ①
43 Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', '
    SubjectCON'))
44
45 %%% iprop!
46 DIRECTORY (data, string) is the directory name where to save the group of
    subjects with connectivity data.
47 %%%% idefault!
48 [fileparts(which('test_brph2')) filesep '
    default_group_subjects_CON_most_likely_to_be_erased']
49
50 %%% iprop!
51 PUT_DIR (query, item) opens a dialog box to set the directory where to save
    the group of subjects with connectivity data.
52 %%%% isettings!
53 'ExporterGroupSubjectCON_XLS'
54 %%%% icalculate!
55 directory = uigetdir('Select directory');
56 if ischar(directory) && isfolder(directory)
57     ex.set('DIRECTORY', directory);
58 end
59 value = ex;
60
61 %%% iprop!
62 SAVE (result, empty) saves the group of subjects with connectivity data in
    XLSX files in the selected directory.
63 %%%% icalculate!
64 directory = ex.get('DIRECTORY');
65
66 if isfolder(directory) ②

```

① Same as in note ① of Code 14.

② Same as in note ④ to ①⑦ in Code 14.


```

67  wb = braph2waitbar(ex.get('WAITBAR'), 0, '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 '
83          num2str(i) ' of ' num2str(sub_number) ' ...'])
84
85      sub = sub_dict.get('IT', i);
86      sub_id = sub.get('ID');
87      sub_CON = sub.get('CON');
88
89      tab = table(sub_CON);
90
91      sub_file = [gr_directory filesep() sub_id '.xlsx'];
92
93      % save file
94      writetable(tab, sub_file, 'WriteVariableNames', false);
95  end
96
97  % variables of interest
98  voi_ids = {};
99  for i = 1:1:sub_number
100      sub = sub_dict.get('IT', i);
101      voi_ids = unique([voi_ids, sub.get('VOI_DICT').get('KEYS')]);
102  end
103  if ~isempty(voi_ids)
104      vois = cell(2 + sub_number, 1 + length(voi_ids));
105      vois{1, 1} = 'Subject ID';
106      vois(1, 2:end) = voi_ids;
107      for i = 1:1:sub_number
108          sub = sub_dict.get('IT', i);
109          vois{2 + i, 1} = sub.get('ID');
110
111          voi_dict = sub.get('VOI_DICT');
112          for v = 1:1:voi_dict.get('LENGTH')
113              voi = voi_dict.get('IT', v);
114              voi_id = voi.get('ID');
115              if isa(voi, 'VOINumeric') % Numeric
116                  vois{2 + i, 1 + find(strcmp(voi_id, voi_ids))} = voi.get
117                      ('V');
118              elseif isa(voi, 'VOICategorical') % Categorical
119                  categories = voi.get('CATEGORIES');
120                  vois{2, 1 + find(strcmp(voi_id, voi_ids))} = cell2str(
121                      categories);
122                  vois{2 + i, 1 + find(strcmp(voi_id, voi_ids))} =
123                      categories{voi.get('V')};
124              end
125          end
126      end
127      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 `_ExporterGroupSubjectCON_XLS.gen.m`. ← [Code 15](#)

```

1 %% itests!
2
3 %%% iexcluded_props!
4 [ExporterGroupSubjectCON_XLS.PUT_DIR]
5
6 %%% itest!
7 %%% iname!
8 Delete directory TBE
9 %%% iprobability!
10 1
11 %%% icode!
12 warning('off', 'MATLAB:DELETE:FileNotFound')
13 dir_to_be_erased = ExporterGroupSubjectCON_XLS.getPropDefault('DIRECTORY');
14 if isfolder(dir_to_be_erased)
15     rmdir(dir_to_be_erased, 's')
16 end
17 warning('on', 'MATLAB:DELETE:FileNotFound')
18
19 %%% itest!
20 %%% iname!
21 Export and import
22 %%% iprobability!
23 .01
24 %%% icode!
25 br1 = BrainRegion( ...
26     'ID', 'ISF', ...
27     'LABEL', 'superiorfrontal', ...
28     'NOTES', 'notes1', ...
29     'X', -12.6, ...
30     'Y', 22.9, ...
31     'Z', 42.4 ...
32 );
33 br2 = BrainRegion( ...
34     'ID', 'lFP', ...
35     'LABEL', 'frontalpole', ...
36     'NOTES', 'notes2', ...
37     'X', -8.6, ...
38     'Y', 61.7, ...
39     'Z', -8.7 ...
40 );
41 br3 = BrainRegion( ...
42     'ID', 'lRMF', ...
43     'LABEL', 'rostralmiddlefrontal', ...
44     'NOTES', 'notes3', ...
45     'X', -31.3, ...
46     'Y', 41.2, ...
47     'Z', 16.5 ...
48 );
49 br4 = BrainRegion( ...
50     'ID', 'lCMF', ...
51     'LABEL', 'caudalmiddlefrontal', ...
52     'NOTES', 'notes4', ...
53     'X', -34.6, ...
54     'Y', 10.2, ...
55     'Z', 42.8 ...
56 );
57 br5 = BrainRegion( ...

```

```

58     'ID', 'lPOB', ...
59     'LABEL', 'parsorbitalis', ...
60     'NOTES', 'notes5', ...
61     'X', -41, ...
62     'Y', 38.8, ...
63     'Z', -11.1 ...
64 );
65
66 ba = BrainAtlas( ...
67     'ID', 'TestToSaveCoolID', ...
68     'LABEL', 'Brain Atlas', ...
69     'NOTES', 'Brain atlas notes', ...
70     'BR_DICT', IndexedDictionary('IT_CLASS', 'BrainRegion', 'IT_LIST', {br1,
71         br2, br3, br4, br5}) ...
72 );
73 sub1 = SubjectCON( ...
74     'ID', 'SUB CON 1', ...
75     'LABEL', 'Subejct CON 1', ...
76     'NOTES', 'Notes on subject CON 1', ...
77     'BA', ba, ...
78     'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
79 );
80 sub1.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 75))
81 sub1.memorize('VOI_DICT').get('ADD', VOICategorical('ID', 'Sex', 'CATEGORIES',
82     {'Female', 'Male'}, 'V', find(strcmp('Female', {'Female', 'Male'}))))
83
84 sub2 = SubjectCON( ...
85     'ID', 'SUB CON 2', ...
86     'LABEL', 'Subejct CON 2', ...
87     'NOTES', 'Notes on subject CON 2', ...
88     'BA', ba, ...
89     'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
90 );
91 sub2.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 70))
92 sub2.memorize('VOI_DICT').get('ADD', VOICategorical('ID', 'Sex', 'CATEGORIES',
93     {'Female', 'Male'}, 'V', find(strcmp('Male', {'Female', 'Male'}))))
94
95 sub3 = SubjectCON( ...
96     'ID', 'SUB CON 3', ...
97     'LABEL', 'Subejct CON 3', ...
98     'NOTES', 'Notes on subject CON 3', ...
99     'BA', ba, ...
100     'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
101 );
102 sub3.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 50))
103 sub3.memorize('VOI_DICT').get('ADD', VOICategorical('ID', 'Sex', 'CATEGORIES',
104     {'Female', 'Male'}, 'V', find(strcmp('Female', {'Female', 'Male'}))))
105
106 gr = Group( ...
107     'ID', 'GR CON', ...
108     'LABEL', 'Group label', ...
109     'NOTES', 'Group notes', ...
110     'SUB_CLASS', 'SubjectCON', ...
111     'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON', 'IT_LIST', {sub1
112         , sub2, sub3}) ...
113 );
114
115 directory = [fileparts(which('test_brph2')) filesep '
116     trial_group_subjects_CON_to_be_erased'];
117 if ~exist(directory, 'dir')

```

```

113     mkdir(directory)
114 end
115
116 ex = ExporterGroupSubjectCON_XLS( ...
117     'DIRECTORY', directory, ...
118     'GR', gr ...
119 );
120 ex.get('SAVE');
121
122 im1 = ImporterGroupSubjectCON_XLS( ...
123     'DIRECTORY', [directory filesep() gr.get(Group.ID)], ...
124     'BA', ba ...
125 );
126 gr_loaded1 = im1.get('GR');
127
128 assert(gr.get('SUB_DICT').get('LENGTH') == gr_loaded1.get('SUB_DICT').get('
    LENGTH'), ...
129 [BRAPH2.STR ':ExporterGroupSubjectCON_XLS:' BRAPH2.FAIL_TEST], ...
130 'Problems saving or loading a group.')
131 for i = 1:1:max(gr.get('SUB_DICT').get('LENGTH'), gr_loaded1.get('SUB_DICT')
    .get('LENGTH'))
132     sub = gr.get('SUB_DICT').get('IT', i);
133     sub_loaded = gr_loaded1.get('SUB_DICT').get('IT', i);
134     assert( ...
135         isequal(sub.get('ID'), sub_loaded.get('ID')) & ...
136         isequal(sub.get('BA'), sub_loaded.get('BA')) & ...
137         isequal(sub.get('VOI_DICT').get('IT', 'Age').get('V'), sub_loaded.
            get('VOI_DICT').get('IT', 'Age').get('V')) & ...
138         isequal(sub.get('VOI_DICT').get('IT', 'Sex').get('V'), sub_loaded.
            get('VOI_DICT').get('IT', 'Sex').get('V')) & ...
139         isequal(sub.get('CON'), sub_loaded.get('CON')), ...
140         [BRAPH2.STR ':ExporterGroupSubjectCON_XLS:' BRAPH2.FAIL_TEST], ...
141         'Problems saving or loading a group.')
142 end
143
144 % import with new brain atlas
145 im2 = ImporterGroupSubjectCON_XLS( ...
146     'DIRECTORY', [directory filesep() gr.get(Group.ID)] ...
147 );
148 gr_loaded2 = im2.get('GR');
149
150 assert(gr.get('SUB_DICT').get('LENGTH') == gr_loaded2.get('SUB_DICT').get('
    LENGTH'), ...
151 [BRAPH2.STR ':ExporterGroupSubjectCON_XLS:' BRAPH2.FAIL_TEST], ...
152 'Problems saving or loading a group.')
153 for i = 1:1:max(gr.get('SUB_DICT').get('LENGTH'), gr_loaded2.get('SUB_DICT')
    .get('LENGTH'))
154     sub = gr.get('SUB_DICT').get('IT', i);
155     sub_loaded = gr_loaded2.get('SUB_DICT').get('IT', i);
156     assert( ...
157         isequal(sub.get('ID'), sub_loaded.get('ID')) & ...
158         ~isequal(sub.get('BA').get('ID'), sub_loaded.get('BA').get('ID')) &
            ...
159         isequal(sub.get('VOI_DICT').get('IT', 'Age').get('V'), sub_loaded.
            get('VOI_DICT').get('IT', 'Age').get('V')) & ...
160         isequal(sub.get('VOI_DICT').get('IT', 'Sex').get('V'), sub_loaded.
            get('VOI_DICT').get('IT', 'Sex').get('V')) & ...
161         isequal(sub.get('CON'), sub_loaded.get('CON')), ...
162         [BRAPH2.STR ':ExporterGroupSubjectCON_XLS:' BRAPH2.FAIL_TEST], ...
163         'Problems saving or loading a group.')
164 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](#)

```

1
2 %% iheader!
3 SubjectFUN < Subject (sub, subject with functional matrix) is a subject with
   functional matrix (e.g. fMRI).
4
5 %%% idescription!
6 Subject with a functional matrix (e.g. obtained from fMRI).
7
8 %%% iseealso!
9 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](#)

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the subject.
5 %%%% idefault!
6 'SubjectFUN'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the subject.
10 %%%% idefault!
11 'Subject with a functional matrix (e.g. obtained from fMRI).'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the subject.
15 %%%% isettings!
16 'SubjectFUN'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the subject.
20 %%%% idefault!
21 'SubjectFUN ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the subject.
25 %%%% idefault!
26 'SubjectFUN label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the subject.
30 %%%% idefault!
31 '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
49           brain regions (' int2str(br_number) ').'];
50 end
51 %%% igui! ②
52 pr = PanelPropMatrix('EL', sub, 'PROP', SubjectFUN.FUN, ...
53     'ROWNAME', {'numbered'}, ...
54     'COLUMNNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
55     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](#)

```

1 %% itests!
2
3 %% itest!
4 %%% iname!
5 GUI
6 %%% iprobability!
7 .01
8 %%% icode!
9 im_ba = ImporterBrainAtlasXLS('FILE', 'aal90_atlas.xlsx');
10 ba = im_ba.get('BA');
11
12 gr = Group('SUB_CLASS', 'SubjectFUN', 'SUB_DICT', IndexedDictionary('
    IT_CLASS', 'SubjectFUN'));
13 for i = 1:1:50
14     sub = SubjectFUN( ...
15         'ID', ['SUB FUN ' int2str(i)], ...
16         'LABEL', ['Subejct FUN ' int2str(i)], ...
17         'NOTES', ['Notes on subject FUN ' int2str(i)], ...
18         'BA', ba, ...
19         'FUN', rand(10, ba.get('BR_DICT').get('LENGTH')) ... ①
20     );
21     sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
        rand()))
22     sub.memorize('VOI_DICT').get('ADD', VOICategorical('ID', 'Sex', '
        CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
23     gr.get('SUB_DICT').get('ADD', sub)
24 end
25
26 gui = GUIElement('PE', gr, 'CLOSEREQ', false);
27 gui.get('DRAW')
28 gui.get('SHOW')
29
30 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](#)

```

1 %% iheader!
2
3 SubjectFUN_MP < Subject (sub, subject with functional multiplex data) is a
  subject with functional multiplex data (e.g. multiplex fMRI).
4
5 %%% idescription!
6 Subject with data for each brain region corresponding to L functional layers
  (e.g. activation timeseries obtained from fMRI or EEG).
7
8 %%% iseealso!
9 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](#)

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the subject.
5 %%% idefault!
6 'SubjectFUN_MP'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the subject.
10 %%% idefault!
11 'Subject with data for each brain region corresponding to L functional
  layers (e.g. activation timeseries obtained from fMRI or EEG).'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the subject.
15 %%% isettings!
16 'SubjectFUN_MP'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the subject.
20 %%% idefault!
21 'SubjectFUN_MP ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the subject.
25 %%% idefault!
26 'SubjectFUN_MP label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the subject.
30 %%% idefault!
31 'SubjectFUN_MP notes'
32

```

```

33 %% iprops!
34
35 %% iprop!
36 BA (data, item) is a brain atlas.
37 %%% isettings!
38 'BrainAtlas'
39
40 %% iprop!
41 L (data, scalar) is the number of layers of subject data. ①
42 %%% idefault!
43 2
44
45 %% iprop!
46 LAYERLABELS (metadata, stringlist) are the layer labels provided by the user
   . ②
47
48 %% iprop!
49 ALAYERLABELS (query, stringlist) returns the processed layer labels. ③
50 %%% icalculate!
51 value = sub.get('LAYERLABELS');
52
53 %% iprop!
54 FUN_MP (data, cell) is a cell containing L matrices with each column
   corresponding to the time series of a brain region.
55 %%% ickvalue!
56 br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
57 num_layers = sub.get('L');
58 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
59 if check
60     msg = 'All ok!';
61 else
62     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) ').'];
63 end
64 %%% igui! ⑤
65 pr = PanelPropCell('EL', sub, 'PROP', SubjectFUN_MP.FUN_MP, ...
66     'TABLE_HEIGHT', s(40), ...
67     'XSLIDERSHOW', true, ...
68     'XSLIDERLABELS', sub.getCallback('ALAYERLABELS'), ...
69     'YSLIDERSHOW', false, ...
70     'ROWNAME', {'numbered'}, ...
71     'COLUMNNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
72     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

```

1 %% itests!
2
3 %% itest!
4 %%% iname!
5 GUI
6 %%% iprobability!
7 .01
8 %%% icode!
9 im_ba = ImporterBrainAtlasXLS('FILE', 'aal90_atlas.xlsx');
10 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](#)

```

1
2 %% iheader!
3 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).
4
5 %%% idescription!
6 Subject with connectivity and functional data (e.g. obtained from DTI and
    fMRI).
7 The first layer contains a connectivity matrix and the second layer contains
    functional data.
8
9 %%% iseealso!
10 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](#)

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the subject.
5 %%% idefault!
6 'SubjectCON_FUN_MP'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the subject.
10 %%% idefault!
11 '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.'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the subject.
15 %%% isettings!
16 'SubjectCON_FUN_MP'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the subject.
20 %%% idefault!
21 'SubjectCON_FUN_MP ID'
22

```

```

23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the subject.
25 %%% idefault!
26 'SubjectCON_FUN_MP label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the subject.
30 %%% idefault!
31 'SubjectCON_FUN_MP notes'
32
33 %%% iprops!
34
35 %%% iprop!
36 BA (data, item) is a brain atlas.
37 %%% isettings!
38 'BrainAtlas'
39
40 %%% iprop!
41 CON (data, smatrix) is an adjacency matrix.
42 %%% icheck_value!
43 br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
44 check = isequal(size(value), [br_number, br_number]); ①
45 if check
46     msg = 'All ok!';
47 else
48     msg = ['CON must be a square matrix with the dimension equal to the
49           number of 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 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](#)

```

1 %% itests!
2
3 %%% itest!
4 %%% iname!
5 GUI
6 %%% iprobability!
7 .01
8 %%% icode!
9 im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx');
10 ba = im_ba.get('BA');
11
12 gr = Group('SUB_CLASS', 'SubjectCON_FUN_MP', 'SUB_DICT', IndexedDictionary('
    IT_CLASS', 'SubjectCON_FUN_MP'));
13 for i = 1:1:50 ①
14     sub = SubjectCON_FUN_MP( ...
15         'ID', ['SUB CON ' int2str(i)], ...
16         'LABEL', ['Subejct CON ' int2str(i)], ...
17         'NOTES', ['Notes on subject CON ' int2str(i)], ...
18         'BA', ba, ...
19         'CON', rand(ba.get('BR_DICT').get('LENGTH')), ... ②
20         'FUN', rand(10, ba.get('BR_DICT').get('LENGTH')) ... ③
21     );
22     sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
        rand()))
23     sub.memorize('VOI_DICT').get('ADD', VOICategorical('ID', 'Sex', '
        CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
24     gr.get('SUB_DICT').get('ADD', sub)
25 end
26
27 gui = GUIElement('PE', gr, 'CLOSEREQ', false);
28 gui.get('DRAW')
29 gui.get('SHOW')
30
31 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](#)

```

1
2 %% iheader!
3 SubjectST < Subject (sub, subject with structural data) is a subject with
   structural data (e.g. sMRI).
4
5 %%% idescription!
6 Subject with structural data (e.g. cortical thickness obtained from
   strcutural MRI) for each brain region.
7
8 %%% iseealso!
9 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](#)

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the subject.
5 %%% idefault!
6 'SubjectST'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the subject.
10 %%% idefault!
11 'SubjectST with structural data (e.g. cortical thickness obtained from
   strcutural MRI) for each brain region.'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the subject.
15 %%% isettings!
16 'SubjectST'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the subject.
20 %%% idefault!
21 'SubjectST ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the subject.
25 %%% idefault!
26 'SubjectST label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the subject.

```

```

30 %%%% idefault!
31 'SubjectST notes'
32
33 %% iprops!
34
35 %%% iprop!
36 BA (data, item) is a brain atlas.
37 %%%% isettings!
38 'BrainAtlas'
39
40 %%% iprop!
41 ST (data, cvector) is a column vector with data for each brain region.
42 %%%% icheck_value!
43 br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
44 check = (iscolumn(value) && isequal(size(value), [br_number, 1])) || (
45     isempty(value) && br_number == 0); ①
46 if check
47     msg = 'All ok!';
48 else
49     msg = ['ST must be a column vector with the same number of element as
50         the brain regions (' int2str(br_number) ').'];
51 end
52 %%%% igui! ②
53 pr = PanelPropMatrix('EL', sub, 'PROP', SubjectST.ST, ...
54     'ROWNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
55     'COLUMNNAME', {}, ...
56     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](#)

```

1 %% itests!
2
3 %% itest!
4 %%% iname!
5 GUI
6 %%% iprobability!
7 .01
8 %%% icode!
9 im_ba = ImporterBrainAtlasXLS('FILE', 'destrieux_atlas.xlsx');
10 ba = im_ba.get('BA');
11
12 gr = Group('SUB_CLASS', 'SubjectST', 'SUB_DICT', IndexedDictionary('IT_CLASS
    ', 'SubjectST'));
13 for i = 1:1:50
14     sub = SubjectST( ...
15         'ID', ['SUB ST ' int2str(i)], ...
16         'LABEL', ['Subejct ST ' int2str(i)], ...
17         'NOTES', ['Notes on subject ST ' int2str(i)], ...
18         'BA', ba, ...
19         'ST', rand(ba.get('BR_DICT').get('LENGTH'), 1) ... ①
20     );
21     sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
        rand()))
22     sub.memorize('VOI_DICT').get('ADD', VOICategorical('ID', 'Sex', '
        CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
23     gr.get('SUB_DICT').get('ADD', sub)
24 end
25
26 gui = GUIElement('PE', gr, 'CLOSEREQ', false);
27 gui.get('DRAW')
28 gui.get('SHOW')
29
30 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](#)

```

1 %% iheader!
2
3 SubjectST_MP < Subject (sub, subject with structural multiplex data) is a
  subject with structural multiplex data (e.g. multiplex sMRI).
4
5 %%% idescription!
6 Subject with data for each brain region corresponding to L structural layers
  (e.g. cortical thickness obtained from structural MRI).
7
8 %%% iseealso!
9 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](#)

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the subject.
5 %%% idefault!
6 'SubjectST_MP'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the subject.
10 %%% idefault!
11 'Subject with data for each brain region corresponding to L structural
  layers (e.g. cortical thickness obtained from structural MRI).'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the subject.
15 %%% isettings!
16 'SubjectST_MP'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the subject.
20 %%% idefault!
21 'SubjectST_MP ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the subject.
25 %%% idefault!
26 'SubjectST_MP label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the subject.
30 %%% idefault!
31 'SubjectST_MP notes'
32

```

```

33 %% iprops!
34
35 %% iprop!
36 BA (data, item) is a brain atlas.
37 %%% isettings!
38 'BrainAtlas'
39
40 %% iprop!
41 L (data, scalar) is the number of layers of subject data. ①
42 %%% idefault!
43 2
44
45 %% iprop!
46 LAYERLABELS (metadata, stringlist) are the layer labels provided by the user
   . ②
47
48 %% iprop!
49 ALAYERLABELS (query, stringlist) returns the processed layer labels. ③
50 %%% icalculate!
51 value = sub.get('LAYERLABELS');
52
53 %% iprop!
54 ST_MP (data, cell) is a cell containing L vectors, each with data for each
   brain region.
55 %%% icalculate!
56 br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
57 num_layers = sub.get('L');
58 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
59 if check
60     msg = 'All ok!';
61 else
62     msg = ['ST_MP must be a column vector with the same number of element as
   the brain regions (' int2str(br_number) ').'];
63 end
64 %%% igui! ⑤
65 pr = PanelPropCell('EL', sub, 'PROP', SubjectST_MP.ST_MP, ...
66     'TABLE-HEIGHT', s(40), ...
67     'XSLIDERSHOW', true, ...
68     'XSLIDERLABELS', sub.getCallback('ALAYERLABELS'), ...
69     'YSLIDERSHOW', false, ...
70     'ROWNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
71     'COLUMNNAME', {}, ...
72     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](#)

```

1
2 %% itests!
3
4 %%% itest!
5 %%% iname!
6 GUI
7 %%% iprobability!
8 .01
9 %%% icode!
10 im_ba = ImporterBrainAtlasXLS('FILE', 'destrieux_atlas.xlsx');

```

```

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

```

① Same as in note ① ② ③ of Code 6.