Implement, Export, and Import Groups of Subjects

The BRAPH 2 Developers

August 14, 2024

This is the developer tutorial for implementing, importing, and exporting groups of subjects. You will learn how to create generator files `*.gen.m` for new subjects and well as to import and export them. All `*.gen.m` files can then be compiled by `braph2genesis`. All types of subjects are extensions of the base element `Subject`. You 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 importers and exporters are extensions of the base elements `Exporter` and `Importer`, respectively. Here, you will use as examples `ImporterGroupSubjectCON_TXT` (importing a group of subjects with connectivity data to a series of TXT file),`ImporterGroupSubjectCON_XLS` (importing a group of subjects with connectivity data to a series of XLSX file), `ExporterGroupSubjectCON_TXT` (exporting a group of subjects with connectivity data to a series of TXT file), and `ExporterGroupSubjectCON_XLS` (exporting 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)

You will start by implementing in detail `SubjectCON`, which holds a connectivity matrix (for example, obtained from DTI data).

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

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

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

%%% ¡seealso! ②
ImporterGroupSubjectFUN_TXT, ExporterGroupSubjectFUN_TXT,
     ImporterGroupSubjectFUN_XLS, ExporterGroupSubjectFUN_XLS

%%% ¡build!
1
```

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

② Other related elements.

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

```
%% ¡props_update!

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

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

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

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

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

```

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

Code 3: **SubjectCON element props.** The `props` section of the generator code in `_SubjectCON.gen.m` defines the properties specific for the `SubjectCON` element, including the connectivity matrix.

```
%% ¡props!

%%% ¡prop! ①
BA (data, item) is a brain atlas.
%%%% ¡settings!
'BrainAtlas'

%%% ¡prop!
CON (data, smatrix) is an adjacency matrix.
%%%% ¡check_value!
br_number = sub.get('BA').get('BR_DICT').get('LENGTH'); ②
check = isequal(size(value), [br_number, br_number]); ③
if check ④
    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
%%%% ¡gui! ⑤
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 brain atlas used for the connectivity matrix.

② determines the number of brain regions from the Brain Atlas.

③ checks that the size of `value` (`value` is the connectivity matrix) is equal to the number of brain regions.

④ returns the check information `msg` according to the variable `check`.

⑤ plots the panel of a property matrix-like with element `sub` and the property number `SubjectCON.CON`. `ROWNAME` and `COLUMNNAME` are the name of the brain regions obtained from brain atlas.

Code 4: **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.

```
%% ¡tests!

%%% ¡test!
%%%% ¡name!
GUI ①
%%%% ¡probability! ②
.01
%%%% ¡code!
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 This test checks that the 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 with random connectivity matrices.

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` switched off the confirmation panel for closing the GUI.

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)

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

Code 5: **SubjectCON_MP element header.** The `header` section of the generator code in `_SubjectCON_MP.gen.m` provides the general information about the `SubjectCON_MP` element. ← Code 1

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

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

%%% ¡seealso!
ImporterGroupSubjectCON_MP_TXT, ExporterGroupSubjectCON_MP_TXT,
     ImporterGroupSubjectCON_MP_XLS, ExporterGroupSubjectCON_MP_XLS

%%% ¡build!
1
```

Code 6: **SubjectCON_MP element props update.** The `props_update` section of the generator code in `_SubjectCON_MP.gen.m` updates the properties of the `Subject` element. ← Code 2

```
%% ¡props_update!

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

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

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

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

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

%%% ¡prop!
```

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

Code 7: **SubjectCON_MP element props.** The `props` section of the generator code in `_SubjectCON_MP.gen.m` defines the properties specific for the `SubjectCON_MP` element, including the connectivity matrices for each layer. ← Code 3

```
%% ¡props!

%%% ¡prop!
BA (data, item) is a brain atlas.
%%%% ¡settings!
'BrainAtlas'

%%% ¡prop!
L (data, scalar) is the number of layers of subject data. (1)
%%%% ¡default!
2 (2)

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

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

%%% ¡prop!
CON_MP (data, cell) is a cell containing L matrices corresponding
     connectivity matrices of each layer.
%%%% ¡check_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
%%%% ¡gui!
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) gets 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) shows the x-axis slider.

(10) shows the x-axis slider's labels.

(11) does not show the y-axis slider.

Code 8: **SubjectCON_MP element tests.** The `tests` section from the element generator `_SubjectCON_MP.gen.m`. ← Code 4

```
%% ¡tests!

%%% ¡test!
%%%% ¡name!
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 using connectivity matrices with size of brain regions by brain regions.

Implementation of importer and exporter (for SubjectCON)

Importer from TXT (ImporterGroupSubjectCON_TXT)

You 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 9: **ImporterGroupSubjectCON_TXT element header.** The `header` section of the generator code in `_ImporterGroupSubjectCON_TXT.gen.m` provides the general information about the `Importer` element.

```
%% ¡header!
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)

%%% ¡description!
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.

%%% ¡seealso!
Group, SunbjectCON, ExporterGroupSubjectCON_TXT

%%% ¡build!
1
```

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

Code 10: **ImporterGroupSubjectCON_TXT element props update.** The `props_update` section of the generator code in `_ImporterGroupSubjectCON_TXT.gen.m` updates the properties of the `Importer` element.

```
%% ¡props_update!

%%% ¡prop!
NAME (constant, string) is the name of the CON subject group importer from
     TXT.
%%%% ¡default!
'ImporterGroupSubjectCON_TXT'

%%% ¡prop!
DESCRIPTION (constant, string) is the description of the CON subject group
     importer from TXT.
%%%% ¡default!
'ImporterGroupSubjectCON_TXT imports a group of subjects with connectivity
     data from a series of TXT file and their covariates (optional) from
     another TXT file.'

```

```
%%% ¡prop!
TEMPLATE (parameter, item) is the template of the CON subject group importer from TXT.
%%%% ¡settings!
'ImporterGroupSubjectCON_TXT'

%%% ¡prop!
ID (data, string) is a few-letter code for the CON subject group importer from TXT.
%%%% ¡default!
'ImporterGroupSubjectCON_TXT ID'

%%% ¡prop!
LABEL (metadata, string) is an extended label of the CON subject group importer from TXT.
%%%% ¡default!
'ImporterGroupSubjectCON_TXT label'

%%% ¡prop!
NOTES (metadata, string) are some specific notes about the CON subject group importer from TXT.
%%%% ¡default!
'ImporterGroupSubjectCON_TXT notes'
```

Code 11: **ImporterGroupSubjectCON_TXT element props.** The `props` section of the generator code in `_ImporterGroupSubjectCON_TXT.gen.m` defines the specific properties of the `ImporterGroupSubjectCON_TXT` element.

```
%% ¡props!

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

%%% ¡prop!
GET_DIR (query, item) opens a dialog box to set the directory from where to load the TXT files of the CON subject group.
%%%% ¡settings!
'ImporterGroupSubjectCON_TXT'
%%%% ¡calculate!
directory = uigetdir('Select directory'); ①
if ischar(directory) && isfolder(directory)
  im.set('DIRECTORY', directory); ②
end
value = im;

%%% ¡prop!
BA (data, item) is a brain atlas.
%%%% ¡settings!
'BrainAtlas'

%%% ¡prop!
GR (result, item) is a group of subjects with connectivity data.
%%%% ¡settings!
'Group'
%%%% ¡check_value!
```

① selects the directory that contains the TXT data.

② saves the directory into the `'DIRECTORY'` property of `im`.

```
check = any(strcmp(value.get(Group.SUB_CLASS_TAG), subclasses('SubjectCON',
     [], [], true))); ③
%%%% ¡default!
Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', '
     SubjectCON')) ④
%%%% ¡calculate! ⑤
gr = Group( ...
    'SUB_CLASS', 'SubjectCON', ...
    'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON') ...
    );

gr.lock('SUB_CLASS'); ⑥

directory = im.get('DIRECTORY'); ⑦
if isfolder(directory) ⑧
    wb = braph2waitbar(im.get('WAITBAR'), 0, 'Reading directory ...'); ⑨

    [~, name] = fileparts(directory); ⑩
    gr.set( ... ⑪
        'ID', name, ...
        'LABEL', name, ...
        'NOTES', ['Group loaded from ' directory] ...
        );

    try
        braph2waitbar(wb, .15, 'Loading subjecy group ...')

        % analyzes directory
        files = dir(fullfile(directory, '*.txt')); ⑫

        if ~isempty(files)
            % brain atlas
            ba = im.get('BA'); ⑬
            if ba.get('BR_DICT').get('LENGTH') == 0 ⑭
                br_number = size(readtable(fullfile(directory, files(1).name
     ), 'Delimiter', '\t'), 1); ⑮
                br_dict = ba.memorize('BR_DICT');
                for j = 1:1:br_number
                    br_dict.get('ADD', BrainRegion('ID', ['br' int2str(j)]))
     ⑯
                end
            end

            % adds subjects
            sub_dict = gr.memorize('SUB_DICT'); ⑰
            for i = 1:1:length(files)
                braph2waitbar(wb, .15 + .85 * i / length(files), ['Loading
     subject ' num2str(i) ' of ' num2str(length(files)) ' ...']) ⑱

    % read file
                [~, sub_id] = fileparts(files(i).name);
                CON = table2array(readtable(fullfile(directory, files(i).
     name), 'Delimiter', '\t')); ⑲
                if size(CON, 1) ~= ba.get('BR_DICT').get('LENGTH') || size(
```

③ checks that the class of subjects of the group is `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.

⑤ constructs an empty `Group`.

⑥ locks the property `'SUB_CLASS'` irreversibly.

⑦ returns the data directory that has been saved at ②.

⑧ checks that directory exists.

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

⑩ extracts the directory name from its complete path.

⑪ sets the properties `'ID'`, `'LABEL'` and `'NOTES'` for the 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 subject to the group.

⑱ updates the waitbar for each file.

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

```
CON, 2) ~= ba.get('BR_DICT').get('LENGTH') 20
                error( ...
                    [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO], ...
                    [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO '\\n' ...
                    'The file ' sub_id ' should contain a matrix ' int2str(ba.get('BR_DICT').get('LENGTH')) 'x' int2str(ba.get('BR_DICT').get('LENGTH')) ', ' ...
                    'while it is ' int2str(size(CON, 1)) 'x' int2str(size(CON, 2)) '.'] ...
                    ) 21
            end

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

        % variables of interest
        if isfile([directory '.vois.txt']) 22
            vois = textread([directory '.vois.txt'], '%s', 'delimiter', '\t', 'whitespace', ''); 23
            vois = reshape(vois, find(strcmp('', vois), 1) - 1, [])'; 24
            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 isempty(vois{2, v}) 25
                        sub.memorize('VOI_DICT').get('ADD', ...
                            VOINumeric( ...
                                'ID', voi_id, ...
                                'V', str2num(vois{i, v}) ...
                                ) ...
                            ); 26
                    elseif ~isempty(vois{2, v}) 27
                        categories = eval(vois{2, v});
                        sub.memorize('VOI_DICT').get('ADD', ...
                            VOICategoric( ...
                                'ID', voi_id, ...
                                'CATEGORIES', str2cell(categories), ...
                                'V', find(strcmp(vois{i, v}, categories)) ...
                                ) ...
                            );
                    end
                end
            end
        end
    end
catch e
    braph2waitbar(wb, 'close')

```

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

21 outputs the error information.

22 adds the variables of interest (vois).

23 reads the file *.vois.txt.

24 reshapes the vois.

25 checks whether the variable is is numeric.

26 adds the variable of interest with 'ID' and value 'V'.

27 checks whether the variable is categorical.

```
        rethrow(e)
    end

  braph2waitbar(wb, 'close') (28)
else
    error([BRAPH2.STR ':ImporterGroupSubjectCON_TXT:' BRAPH2.ERR_IO], ...
        [BRAPH2.STR ':ImporterGroupSubjectCON_TXT:' BRAPH2.ERR_IO '\\n' ...
        'The prop DIRECTORY must be an existing directory, but it is '''
     directory '''.'] ...
        );
end

value = gr;
```

(28) closes the waitbar.

Code 12: **ImporterGroupSubjectCON_TXT element tests.** The tests section from the element generator _ImporterGroupSubjectCON_TXT.gen.m. In this section, some example data are created for testing.

```
%% ¡tests!

%%% ¡excluded_props! (1)
[ImporterGroupSubjectCON_TXT.GET_DIR]

%%% ¡test!
%%%% ¡name!
Create example files (2)
%%%% ¡code!
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)

    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)
```

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

(2) creates the example files.

(3) defines the directory `'Example data CON TXT'` where the example data will be contained.

(4) creates the directory for the example data.

(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 2) for group 1.

(11) assigns the rewiring probability for group 1.

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

```
        {{'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

        writetable(array2table(A1), [gr1_dir filesep() sub_id '.txt'], '
 Delimiter', '\t', 'WriteVariableNames', false) ⑱

        vois1 = [vois1; {sub_id, randi(90), sex_options(randi(2))}]; ⑲
    end
    writetable(table(vois1), [data_dir filesep() gr1_name '.vois.txt'], '
 Delimiter', '\t', 'WriteVariableNames', false) ⑳

    % Group 2 ㉑
    K2 = 2;
    beta2 = 0.85;
    gr2_name = 'CON_Group_2_TXT';
    gr2_dir = [data_dir filesep() gr2_name];
    mkdir(gr2_dir);
    vois2 = [
        {{'Subject ID'} {'Age'} {'Sex'}}
        {{} {} {['{' sprintf(' ''%s'' ', sex_options{:}) '}']}}
        ];
    for i = 51:1:100
        sub_id = ['SubjectCON_MP_' num2str(i)];

        h2 = WattsStrogatz(N, K2, beta2);

        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() 'SubjectCON_' num2str
 (i) '.txt'], 'Delimiter', '\t', '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.txt'], '
 Delimiter', '\t', 'WriteVariableNames', false)

    % reset RNG
    rng(rng_settings_) ㉒
end

%%% ¡test_functions!
```

⑬ generates 50 subjects.

⑭ creates a Watts-Strogatz graph.

⑮ extracts the adjacency matrix.

⑯ makes the adjacency matrix weighted.

⑰ makes the adjacency matrix symmetric.

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.

```
function h = WattsStrogatz(N,K,beta) ㉓
% 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); ㉔
t = s + repmat(1:K,N,1);  ㉕
t = mod(t-1,N)+1; ㉖

for source=1:N ㉗
    switchEdge = rand(K, 1) < beta; ㉘

    newTargets = rand(N, 1); ㉙
    newTargets(source) = 0;
    newTargets(s(t==source)) = 0;
    newTargets(t(source, ~switchEdge)) = 0;

    [~, ind] = sort(newTargets, 'descend');
    t(source, switchEdge) = ind(1:nnz(switchEdge)); ㉚
end

h = graph(s,t); ㉛
end
```

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`.

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`.

```
%%% ¡test!
%%%% ¡name!
GUI
%%%% ¡probability!
.01
%%%% ¡code!
im_ba = ImporterBrainAtlasTXT('FILE', [fileparts(which('SubjectCON'))
     filesep 'Example data CON TXT' filesep 'atlas.txt']);
ba = im_ba.get('BA');

im_gr = ImporterGroupSubjectCON_TXT( ... ㉜
    'DIRECTORY', [fileparts(which('SubjectCON')) filesep 'Example data CON
     TXT' filesep 'CON_Group_1_TXT'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );
gr = im_gr.get('GR'); ㉝

gui = GUIElement('PE', gr, 'CLOSEREQ', false); ㉞
gui.get('DRAW')
gui.get('SHOW')

gui.get('CLOSE')
```

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 without requiring close confirmation.

Importer from XLS/XLSX (ImporterGroupSubjectCON_XLS)

You will now see how to implement in detail `ImporterGroupSubjectCON_XLS` modifying `ImporterGroupSubjectCON_TXT`. The data should be stored in the folders `Group1` and `Group2`, and the file format is `.xls` or `.xlsx`.

Code 13: **ImporterGroupSubjectCON_XLS element header.** The `header` section of the generator code in `_ImporterGroupSubjectCON_XLS.gen.m` provides the general information about the `Importer` element. ← Code 9

```
%% ¡header!
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.

%%% ¡description!
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.

%%% ¡seealso!
Group, SubjectCON, ExporterGroupSubjectCON_XLS

%%% ¡build!
1
```

Code 14: **ImporterGroupSubjectCON_XLS element props update.** The `props_update` section of the generator code in `_ImporterGroupSubjectCON_XLS.gen.m` updates the properties of the `Importer` element. ← Code 10

```
%% ¡props_update!

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

%%% ¡prop!
DESCRIPTION (constant, string) is the description of the CON subject group
     importer from XLS/XLSX.
%%%% ¡default!
'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.'

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

```
%%%% ¡settings!
'ImporterGroupSubjectCON_XLS'

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

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

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

Code 15: **ImporterGroupSubjectCON_XLS element props.** The `props` section of the generator code in `_ImporterGroupSubjectCON_XLS.gen.m` defined the properties specific for `ImporterGroupSubjectCON_XLS`. ← Code 10

```
%% ¡props!

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

%%% ¡prop!
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.
%%%% ¡settings!
'ImporterGroupSubjectCON_XLS'
%%%% ¡calculate!
directory = uigetdir('Select directory');
if ischar(directory) && isfolder(directory)
    im.set('DIRECTORY', directory);
end
value = im;

%%% ¡prop!
BA (data, item) is a brain atlas.
%%%% ¡settings!
'BrainAtlas'

%%% ¡prop!
GR (result, item) is a group of subjects with connectivity data.
%%%% ¡settings!
'Group'
%%%% ¡check_value!
check = any(strcmp(value.get(Group.SUB_CLASS_TAG), subclasses('SubjectCON',
     [], [], true))); ①
%%%% ¡default!
```

① Same as in note ③ of Code 10.

```
Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON')) [2]
%%%% ¡calculate! [3]
gr = Group( ...
    'SUB_CLASS', 'SubjectCON', ...
    'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON') ...
    );

gr.lock('SUB_CLASS');

directory = im.get('DIRECTORY');
if isfolder(directory)
    wb = braph2waitbar(im.get('WAITBAR'), 0, 'Reading directory ...');

    [~, gr_name] = fileparts(directory);
    gr.set( ...
        'ID', gr_name, ...
        'LABEL', gr_name, ...
        'NOTES', ['Group loaded from ' directory] ...
        );

    try
        braph2waitbar(wb, .15, 'Loading subject group ...')

        % analyzes directory
        files = [dir(fullfile(directory, '*.xlsx')); dir(fullfile(directory, '*.xls'))];

        if ~isempty(files)
            % brain atlas
            ba = im.get('BA');
            if ba.get('BR_DICT').get('LENGTH') == 0
                br_number = size(xlsread(fullfile(directory, files(1).name)), 1);
                br_dict = ba.memorize('BR_DICT');
                for j = 1:1:br_number
                    br_dict.get('ADD', BrainRegion('ID', ['br' int2str(j)]))
                end
            end

            % adds subjects
            sub_dict = gr.memorize('SUB_DICT');
            for i = 1:1:length(files)
                braph2waitbar(wb, .15 + .85 * i / length(files), ['Loading subject ' num2str(i) ' of ' num2str(length(files)) ' ...'])

                % read file
                [~, sub_id] = fileparts(files(i).name);

                CON = xlsread(fullfile(directory, files(i).name));
                if size(CON, 1) ~= ba.get('BR_DICT').get('LENGTH') || size(CON, 2) ~= ba.get('BR_DICT').get('LENGTH')
                    error( ...
                        [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO], ...
                        [BRAPH2.STR ':' class(im) ':' BRAPH2.ERR_IO '\\n' ...
                        'The file ' sub_id ' should contain a matrix ' int2str(ba.get('BR_DICT').get('LENGTH')) 'x' int2str(ba.get('BR_DICT').get('LENGTH')) ', ' ...
                        'while it is ' int2str(size(CON, 1)) 'x' int2str(size(CON, 2)) '.'] ...
```

[2] Same as in note [4] of Code 10.

[3] Same as in note [5] to [28] in Code 10.

```
                        )
                end

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

```
%% ¡tests!

%%% ¡excluded_props!
[ImporterGroupSubjectCON_XLS.GET_DIR]

%%% ¡test!
%%%% ¡name!
Create example files
%%%% ¡code!
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

%%% ¡test_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;

    [~, ind] = sort(newTargets, 'descend');
    t(source, switchEdge) = ind(1:nnz(switchEdge));
end

h = graph(s,t);
end

%%% ¡test!
%%%% ¡name!
GUI
%%%% ¡probability!
.01
%%%% ¡code!
im_ba = ImporterBrainAtlasXLS('FILE', [fileparts(which('SubjectCON'))
     filesep 'Example data CON XLS' filesep 'atlas.xlsx']);
ba = im_ba.get('BA');

im_gr = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('SubjectCON')) filesep 'Example data CON
     XLS' filesep 'CON_Group_1_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );
gr = im_gr.get('GR');

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

gui.get('CLOSE')
```

Exporter to TXT (ExporterGroupSubjectCON_TXT)

In this section, you will see how to implement in detail `ExporterGroupSubjectCON_TXT`. The data will be stored in the folders `Group1` and `Group2`, and the file format is `*.txt`.

Code 17: **ExporterGroupSubjectCON_TXT element header.** The `header` section of the generator code in `_ExporterGroupSubjectCON_TXT.gen.m` provides the general information about the `ExporterGroupSubjectCON_TXT` element.

```
%% ¡header!
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)

%%% ¡description!
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.

%%% ¡seealso!
Group, SunbjectCON, ExporterGroupSubjectCON_TXT

%%% ¡bluid!
1
```

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

Code 18: **ExporterGroupSubjectCON_TXT element props update.** The `props_update` section of the generator code in `_ExporterGroupSubjectCON_TXT.gen.m` updates the properties of the `Exporter` element.

```
%% ¡props_update!

%%% ¡prop!
NAME (constant, string) is the name of the CON subject group exporter in TXT
     .
%%%% ¡default!
'ExporterGroupSubjectCON_TXT'

%%% ¡prop!
DESCRIPTION (constant, string) is the description of the CON subject group
     exporter in TXT.
%%%% ¡default!
'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.'

%%% ¡prop!
TEMPLATE (parameter, item) is the template of the CON subject group exporter
      in TXT.
```

```
%%%% ¡settings!
'ExporterGroupSubjectCON_TXT'

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

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

%%% ¡prop!
NOTES (metadata, string) are some specific notes about the CON subject group exporter in TXT.
%%%% ¡default!
'ExporterGroupSubjectCON_TXT notes'
```

Code 19: **ExporterGroupSubjectCON_TXT element props.** The props section of the generator code in _ExporterGroupSubjectCON_TXT.gen.m defines the properties specific for the ExporterGroupSubjectCON_TXT element.

```
%% ¡props!

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

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

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

%%% ¡prop!
```

① checks that the SUB_CLASS_TAG is equal to 'SubjectCON'.

② defines the export directory.

③ selects the export directory.

④ checks the export directory before setting it.

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

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

    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) '...']) ⑦

        sub = sub_dict.get('IT', i); ⑧
        sub_id = sub.get('ID'); ⑨
        sub_CON = sub.get('CON'); ⑩

        tab = table(sub_CON); ⑪

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

        % save file
        writetable(tab, sub_file, 'Delimiter', '\t', 'WriteVariableNames',
     false); ⑫
    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')]); ⑬
    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') ⑭
                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
```

⑤ checks the export directory is a folder.
⑥ creates the waitbar with an initial progress of `0`. Displaying the character `'Retrieving path ...'`.

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.

```
    ('V');
            elseif isa(voi, 'VOICategoric') % Categoric
                categories = voi.get('CATEGORIES');
                vois{2, 1 + find(strcmp(voi_id, voi_ids))} = {['{' sprintf(' ''%s'' ', categories{:}) '}']};
                vois{2 + i, 1 + find(strcmp(voi_id, voi_ids))} = categories{voi.get('V')};
            end
        end
    end
    writetable(table(vois), [gr_directory '.vois.txt'], 'Delimiter', '\t', 'WriteVariableNames', false) (15)
end

braph2waitbar(wb, 'close') (16)
end
value = [];
```

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

(16) closes the waitbar.

Code 20: **ExporterGroupSubjectCON_TXT element tests.** The `tests` section from the element generator `_ExporterGroupSubjectCON_TXT.gen.m`.

```
%% ¡tests!

%%% ¡excluded_props! (1)
[ExporterGroupSubjectCON_TXT.PUT_DIR]

%%% ¡test!
%%%% ¡name!
Delete directory TBE (2)
%%%% ¡probability!
1
%%%% ¡code!
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')

%%% ¡test!
%%%% ¡name!
Export and import (3)
%%%% ¡probability!
.01
%%%% ¡code!
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', ...
```

(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'`.

```
    '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}) ... 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'}))))
```

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'`.

```
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
    );

directory = [fileparts(which('test_braph2')) filesep 'trial_group_subjects_CON_to_be_erased']; ⑩
if ~exist(directory, 'dir')
    mkdir(directory)
end

ex = ExporterGroupSubjectCON_TXT( ... ⑪
    'DIRECTORY', directory, ...
    'GR', gr ...
    );
ex.get('SAVE');

im1 = ImporterGroupSubjectCON_TXT( ... ⑫
    '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_TXT:' BRAPH2.FAIL_TEST], ...
    'Problems saving or loading a group.') ⑬
for i = 1:1:max(gr.get('SUB_DICT').get('LENGTH'), gr_loaded1.get('SUB_DICT').get('LENGTH')) ⑭
    sub = gr.get('SUB_DICT').get('IT', i);
    sub_loaded = gr_loaded1.get('SUB_DICT').get('IT', i);
    assert( ...⑮
        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(round(sub.get('CON'), 10), round(sub_loaded.get('CON'), 10)), ...
        [BRAPH2.STR ':ExporterGroupSubjectCON_TXT:' BRAPH2.FAIL_TEST], ...
        'Problems saving or loading a group.')
```

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

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.

```
end


im2 = ImporterGroupSubjectCON_TXT( ... 16
    '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_TXT:' 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(round(sub.get('CON'), 10), round(sub_loaded.get('CON'), 10)), ...
        [BRAPH2.STR ':ExporterGroupSubjectCON_TXT:' BRAPH2.FAIL_TEST], ...
        'Problems saving or loading a group.')
end

rmdir(directory, 's') 17
```

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

17 deletes the testing data.

Exporter to XLS/XLSX (ExporterGroupSubjectCON_XLS)

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

Code 21: **ExporterGroupSubjectCON_XLS element header.** The `header` section of the generator code in `_ExporterGroupSubjectCON_XLS.gen.m` provides the general information about the `Exporter` element. ← Code 17

```
%% ¡header!
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.

%%% ¡description!
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.

%%% ¡seealso!
Group, SunbjectCON, ImporterGroupSubjectCON_XLS

%%% ¡build!
1
```

Code 22: **ExporterGroupSubjectCON_XLS element props update.** The `props_update` section of the generator code in `_ExporterGroupSubjectCON_XLS.gen.m` updates the properties of the `Exporter` element. ← Code 18

```
%% ¡props_update!

%%% ¡prop!
NAME (constant, string) is the name of the CON subject group exporter in
     XLSX.
%%%% ¡default!
'ExporterGroupSubjectCON_XLS'

%%% ¡prop!
DESCRIPTION (constant, string) is the description of the CON subject group
     exporter in XLSX.
%%%% ¡default!
'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.'

%%% ¡prop!
TEMPLATE (parameter, item) is the template of the CON subject group exporter
      in XLSX.
%%%% ¡settings!
```

```
'ExporterGroupSubjectCON_XLS'

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

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

%%% ¡prop!
NOTES (metadata, string) are some specific notes about the CON subject group
      exporter in XLSX.
%%%% ¡default!
'ExporterGroupSubjectCON_XLS notes'
```

Code 23: **ExporterGroupSubjectCON_XLS element props.** The `props` section of the generator code in `_ExporterGroupSubjectCON_XLS.gen.m` defines the properties specific for the `ExporterGroupSubjectCON_XLS` element. ← Code 19

```
%% ¡props!

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

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

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

%%% ¡prop!
```

(1) Same as in note (1) of Code 18.

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

if isfolder(directory) [2]
    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);
```

[2] Same as in note [4] to [17] in Code 18.

```
                    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)
    end

    braph2waitbar(wb, 'close')
end

value = [];
```

Code 24: **ExporterGroupSubjectCON_XLS element tests.** The `tests` section from the element generator `_ExporterGroupSubjectCON_XLS.gen.m`. ← Code 20

```
%% ¡tests!

%%% ¡excluded_props!
[ExporterGroupSubjectCON_XLS.PUT_DIR]

%%% ¡test!
%%%% ¡name!
Delete directory TBE
%%%% ¡probability!
1
%%%% ¡code!
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')

%%% ¡test!
%%%% ¡name!
Export and import
%%%% ¡probability!
.01
%%%% ¡code!
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

rmdir(directory, 's')
```

Implementation of a subject with functional data

Subject with functional data (SubjectFUN)

In this section, you will see how to implement in detail `SubjectFUN`, which holds timeseries data such as those obtained from fMRI.

Code 25: **SubjectFUN element header.** The `header` section of the generator code in `_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

%%% ¡build!
1
```

Code 26: **SubjectFUN element props update.** The `props_update` section of the generator code in `_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'
```

Code 27: **SubjectFUN element props.** The `props` section of the generator code in `_SubjectFUN.gen.m` defines the properties specific for the `SubjectFUN` element. ← Code 3

```
%% ¡props!

%%% ¡prop!
BA (data, item) is a brain atlas.
%%%% ¡settings!
'BrainAtlas'

%%% ¡prop!
FUN (data, matrix) is an adjacency matrix.
%%%% ¡check_value!
br_number = sub.get('BA').get('BR_DICT').get('LENGTH');
check = size(value, 2) == br_number; ①
if check
    msg = 'All ok!';
else
    msg = ['FUN must be a matrix with the same number of columns as the
     brain regions (' int2str(br_number) ').'];
end
%%%% ¡gui! ②
pr = PanelPropMatrix('EL', sub, 'PROP', SubjectFUN.FUN, ...
    'ROWNAME', {'numbered'}, ...
    'COLUMNNAME', sub.get('BA').get('BR_DICT').getCallback('KEYS'), ...
    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 28: **SubjectFUN element tests.** The `tests` section from the element generator `_SubjectFUN.gen.m`. ← Code 4

```
%% ¡tests!

%%% ¡test!
%%%% ¡name!
GUI
%%%% ¡probability!
.01
%%%% ¡code!
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)))
```

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

```
    gr.get('SUB_DICT').get('ADD', sub)
end

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

gui.get('CLOSE')
```

Subject with functional multiplex data (SubjectFUN_MP)

In this section, you will see how to implement in detail `SubjectFUN_MP`, which can hold, for example, multilayer fMRI data.

Code 29: **SubjectFUN_MP element header.** The `header` section of the generator code in `_SubjectFUN_MP.gen.m` provides the general information about the `SubjectFUN_MP` element. ← Code 5

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

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

%%% ¡seealso!
ImporterGroupSubjectFUN_MP_TXT, ExporterGroupSubjectFUN_MP_TXT,
     ImporterGroupSubjectFUN_MP_XLS, ExporterGroupSubjectFUN_MP_XLS

%%% ¡build!
1
```

Code 30: **SubjectFUN_MP element props update.** The `props_update` section of the generator code in `_SubjectFUN_MP.gen.m` updates the properties of the `Subject` element. ← Code 6

```
%% ¡props_update!

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

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

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

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

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

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

```
'SubjectFUN_MP notes'
```

Code 31: **SubjectFUN_MP element props.** The `props` section of the generator code in `_SubjectFUN_MP.gen.m` defines the properties specific for the `SubjectFUN_MP` element. ← Code 7

```
%% ¡props!

%%% ¡prop!
BA (data, item) is a brain atlas.
%%%% ¡settings!
'BrainAtlas'

%%% ¡prop!
L (data, scalar) is the number of layers of subject data. ①
%%%% ¡default!
2

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

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

%%% ¡prop!
FUN_MP (data, cell) is a cell containing L matrices with each column
     corresponding to the time series of a brain region.
%%%% ¡check_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
%%%% ¡gui! 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 6.

② Same as in note ② of Code 6.

③ Same as in note ③ of Code 6.

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 notes ⑧—⑪ of Code 6.

Code 32: **SubjectFUN_MP element tests.** The `tests` section from the element generator `_SubjectFUN_MP.gen.m`. ← Code 8

```
%% ¡tests!

%%% ¡test!
```

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

gr = Group('SUB_CLASS', 'SubjectFUN_MP', 'SUB_DICT', IndexedDictionary('
     IT_CLASS', 'SubjectFUN_MP'));
for i = 1:1:10 (1)
    sub = SubjectFUN_MP( ...
        'ID', ['SUB FUN_MP ' int2str(i)], ...
        'LABEL', ['Subejct FUN_MP ' int2str(i)], ...
        'NOTES', ['Notes on subject FUN_MP ' int2str(i)], ...
        'BA', ba, ...
        'L', 3, ...
        'LAYERLABELS', {'L1' 'L2' 'L3'}, ...
        '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')
     )} ...
        );
    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 notes (1)–(3) of Code 8.

Implementation of a subject with connectivity and functional data

Subject with connectivity and functional multiplex data (SubjectCON_FUN_MP)

In this section, you will see how to implement `SubjectCON_FUN_MP`. For example, the connectivity data can be obtained from DTI and the functional data can be obtained from fMRI.

Code 33: **SubjectCON_FUN_MP element header.** The `header` section of the generator code in `_SubjectCON_FUN_MP.gen.m` provides the general information about the `SubjectCON_FUN_MP` element. ← Code 5

```
%% ¡header!
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).

%%% ¡description!
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.

%%% ¡seealso!
CombineGroups_CON_FUN_MP, SeparateGroups_CON_FUN_MP

%%% ¡build!
1
```

Code 34: **SubjectCON_FUN_MP element props update.** The `props_update` section of the generator code in `_SubjectCON_FUN_MP.gen.m` updates the properties of the `Subject` element. ← Code 6

```
%% ¡props_update!

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

%%% ¡prop!
DESCRIPTION (constant, string) is the description of the subject.
%%%% ¡default!
'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.'

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

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

```
'SubjectCON_FUN_MP ID'

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

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

Code 35: **SubjectCON_FUN_MP element props.** The `props` section of the generator code in `_SubjectCON_FUN_MP.gen.m` defines the properties specific for the `Subject_FUN_MP` element. ← Code 7

```
%% ¡props!

%%% ¡prop!
BA (data, item) is a brain atlas.
%%%% ¡settings!
'BrainAtlas'

%%% ¡prop!
CON (data, smatrix) is an adjacency matrix.
%%%% ¡check_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
%%%% ¡gui! 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 6.

② Same as in note ④ of Code 6.

Code 36: **SubjectCON_FUN_MP element tests.** The `tests` section from the element generator `_SubjectCON_FUN_MP.gen.m`. ← Code 8

```
%% ¡tests!

%%% ¡test!
%%%% ¡name!
GUI
%%%% ¡probability!
.01
%%%% ¡code!
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( ...
```

① Same as in note ⑥ ⑦ of Code 4.

```
        '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')
```

② constructs connectivity matrix.

③ constructs functional matrix.

Implementation of a subject with structural data

Subject with structural data (SubjectST)

In this section, you will see how to implement `SubjectST`. For example, the structural data can be obtained from sMRI.

Code 37: **SubjectST element header.** The `header` section of the generator code in `_SubjectST.gen.m` provides the general information about the `SubjectST` element. ← Code 1

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

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

%%% ¡seealso!
ImporterGroupSubjectST_TXT, ExporterGroupSubjectST_TXT,
    ImporterGroupSubjectST_XLS, ExporterGroupSubjectST_XLS

%%% ¡build!
1
```

Code 38: **SubjectST element props update.** The `props_update` section of the generator code in `_SubjectST.gen.m` updates the properties of the `Subject` element. ← Code 2

```
%% ¡props_update!

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

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

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

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

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

```

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

Code 39: **SubjectST element props.** The `props` section of the generator code in `_SubjectST.gen.m` defines the properties specific for the `SubjectST` element. ← Code 2

```
%% ¡props!

%%% ¡prop!
BA (data, item) is a brain atlas.
%%%% ¡settings!
'BrainAtlas'

%%% ¡prop!
ST (data, cvector) is a column vector with data for each brain region.
%%%% ¡check_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
%%%% ¡gui! 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 40: **SubjectST element tests.** The `tests` section from the element generator `_SubjectST.gen.m`. ← Code 4

```
%% ¡tests!

%%% ¡test!
%%%% ¡name!
GUI
%%%% ¡probability!
.01
%%%% ¡code!
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
        );
```

① constructs the random adjacency matrix with size of the number of brain regions by 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')
```

Subject with structural multiplex data (SubjectST_MP)

In this section, you will see how to implement in detail `SubjectST_MP`. For example, the structural data can be obtained from multiple sMRI.

Code 41: **SubjectST_MP element header.** The `header` section of the generator code in `_SubjectST_MP.gen.m` provides the general information about the `SubjectST_MP` element.← Code 5

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

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

%%% ¡seealso!
ImporterGroupSubjectST_MP_TXT, ExporterGroupSubjectST_MP_TXT,
     ImporterGroupSubjectST_MP_XLS, ExporterGroupSubjectST_MP_XLS

%%% ¡build!
1
```

Code 42: **SubjectST_MP element props update.** The `props_update` section of the generator code in `_SubjectST_MP.gen.m` updates the properties of the `Subject` element. ← Code 6

```
%% ¡props_update!

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

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

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

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

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

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

```
'SubjectST_MP notes'
```

Code 43: **SubjectST_MP element props.** The `props` section of the generator code in `_SubjectST_MP.gen.m` defines the properties specific for the `SubjectST_MP` element. ← Code 7

```
%% ¡props!

%%% ¡prop!
BA (data, item) is a brain atlas.
%%%% ¡settings!
'BrainAtlas'

%%% ¡prop!
L (data, scalar) is the number of layers of subject data. ①
%%%% ¡default!
2

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

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

%%% ¡prop!
ST_MP (data, cell) is a cell containing L vectors, each with data for each
     brain region.
%%%% ¡check_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
%%%% ¡gui! 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 6.

② Same as in note ② of Code 6.

③ Same as in note ③ of Code 6.

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 6.

Code 44: **SubjectST_MP element tests.** The `tests` section from the element generator `_SubjectST_MP.gen.m`. ← Code 8

```
%% ¡tests!

%%% ¡test!
```

```
%%%% ¡name!
GUI
%%%% ¡probability!
.01
%%%% ¡code!
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 8.