Implement a new Property Panel

The BRAPH 2 Developers

September 21, 2023

This is the developer tutorial for implementing a new figure panel. In this tutorial, we will explain how to create the generator file `*.gen.m` for a new figure panel, which can then be compiled by `braph2genesis`. All figure panels are (direct or indirect) extensions of the element `PanelFig`. We will use the figure panels `BrainSurfacePF` and `BrainAtlasPF` as an examples.

Contents

Implementation of Figure Panel (BrainSurfacePF)

To illustrate the general concepts of a figure panel, we will start by implementing in detail the figure panel `BrainSurfacePF`, which is a direct extension of the element `PanelFig`.

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

```
%% ¡header!
BrainSurfacePF < PanelFig (pf, panel figure brain surface) is a plot of a
    brain surfce. (1)

%%% ¡description!
BrainSurfacePF manages the plot of the brain surface choosen by the user.
A collection of brain surfaces in NV format can be found in the folder
./braph2/brainsurfs/.
This class provides the common methods needed to manage the plot of
the surface. In particular, the user can change lighting, material,
camlight, shadning, colormap, facecolor, brain color, face color,
edge color, and background color.

%%% ¡seealso!
BrainSurface
```

(1) The element `BrainSurfacePF` is defined as a subclass of `PabelFig`. The moniker will be `pf`.

Code 2: **BrainSurfacePF element constants.** The `constants` section of the generator code for `_BrainSurfacePF.gen.m` introductes some element constants. These will simplify the management of the visualization of the brain surface.

```
%% ¡constants!

% fixed 3d view
VIEW_3D = 1 % 3D view numeric code
VIEW_3D_CMD = '3D' % 3D view name
VIEW_3D_AZEL = [-37.5 30] % 3D view azimutal and polar angles

% sagittal left view
VIEW_SL = 2 % sagittal left view numeric code
VIEW_SL_CMD = 'Sagittal left' % sagittal left view name
VIEW_SL_AZEL = [-90 0] % sagittal left view azimutal and polar angles

% sagittal right view
VIEW_SR = 3 % sagittal right view numeric code
VIEW_SR_CMD = 'Sagittal right' % sagittal right view name
VIEW_SR_AZEL = [90 0] % sagittal right view azimutal and polar angles

% axial dorsal view
VIEW_AD = 4 % axial dorsal view numeric code
VIEW_AD_CMD = 'Axial dorsal' % axial dorsal view name
VIEW_AD_AZEL = [0 90] % axial dorsal view azimutal and polar angles

% axial ventral view
VIEW_AV = 5 % axial ventral view numeric code
VIEW_AV_CMD = 'Axial ventral' % axial ventral view name
VIEW_AV_AZEL = [0 -90] % axial ventral view azimutal and polar angles

```

```
% coronal anterior view
VIEW_CA = 6 % coronal anterior view numeric code
VIEW_CA_CMD = 'Coronal anterior' % coronal anterior view name
VIEW_CA_AZEL = [180 0] % coronal anterior view azimutal and polar angles

% coronal posterior view
VIEW_CP = 7 % coronal posterior view numeric code
VIEW_CP_CMD = 'Coronal posterior' % coronal posterior view name
VIEW_CP_AZEL = [0 0] % coronal posterior view azimutal and polar angles

VIEW_CMD = { ... % vector of view names
    BrainSurfacePF.VIEW_3D_CMD ...
    BrainSurfacePF.VIEW_SL_CMD ...
    BrainSurfacePF.VIEW_SR_CMD ...
    BrainSurfacePF.VIEW_AD_CMD ...
    BrainSurfacePF.VIEW_AV_CMD ...
    BrainSurfacePF.VIEW_CA_CMD ...
    BrainSurfacePF.VIEW_CP_CMD ...
    }

VIEW_AZEL = { ... % vector of view azimutal and polar angle
    BrainSurfacePF.VIEW_3D_AZEL ...
    BrainSurfacePF.VIEW_SL_AZEL ...
    BrainSurfacePF.VIEW_SR_AZEL ...
    BrainSurfacePF.VIEW_AD_AZEL ...
    BrainSurfacePF.VIEW_AV_AZEL ...
    BrainSurfacePF.VIEW_CA_AZEL ...
    BrainSurfacePF.VIEW_CP_AZEL ...
    }
```

Code 3: **BrainSurfacePF element new props.** The `props` section of the generator code for `_BrainSurfacePF.gen.m` defines the necessary user interface objects and their callbacks.

```
%% ¡props!

%%% ¡prop!
H_AXES (evanescent, handle) is the handle for the axes. ①
%%%% ¡calculate!
h_axes = uiaxes( ...
    'Parent', pf.memorize('H'), ... ②
    'Tag', 'H_AXES', ...
    'Units', 'normalized', ...
    'OuterPosition', [0 0 1 1] ...
    );
h_axes.Toolbar.Visible = 'off';
h_axes.Interactions = [];
value = h_axes;

%%% ¡prop!
VIEW (figure, rvector) sets the desired view as the line-of-sight azimuth
     and elevation angles. ③
%%%% ¡check_prop!
check = length(value) == 2;
%%%% ¡default!
BrainSurfacePF.VIEW_SL_AZEL
%%%% ¡postset! ④
if pf.get('DRAWN')
    view(pf.get('H_AXES'), pf.get('VIEW'))
```

① defines the evanescent handle of the axes where the brain surface will be plotted. It also defines its general properties.

② ensures that the parent panel is memorized.

③ determines the view of the brain surface.

④ is executed only when the `VIEW` property is set. It takes care of adjusting the view and resetting the lightning.

```

    % reset the ambient lighting
    pf.memorize('ST_AMBIENT').set('PANEL', pf, 'PROP', pf.H_AXES).get('SETUP')
end
%%%% ¡gui!
pr = PanelPropRVectorView('EL', pf, 'PROP', BrainSurfacePF.VIEW, varargin{:});

%%% ¡prop!
ST_AXIS (figure, item) determines the axis settings. ⑤
%%%% ¡settings!
'SettingsAxis'
%%%% ¡default!
SettingsAxis('GRID', false, 'AXIS', false) ⑥
%%%% ¡gui! ⑦
pr = SettingsAxisPP('EL', pf, 'PROP', BrainSurfacePF.ST_AXIS, varargin{:});

%%% ¡prop!
SURFFILE (figure, option) is the name of the file of the brain surface to be plotted. ⑧
%%%% ¡settings!
{dir([fileparts(which('braph2')) filesep() 'brainsurfs' filesep() '*.nv']).name}
%%%% ¡default!
'human_ICBM152.nv'
%%%% ¡postset! ⑨
bs = ImporterBrainSurfaceNV('FILE', pf.get('SURFFILE')).get('SURF');
pf.set('SURF', bs)

if pf.get('DRAWN')
    delete(pf.get('H_BRAIN'))
    pf.set('H_BRAIN', Element.getNoValue())

    pf.memorize('H_BRAIN')

    pf.set('BRAIN', pf.get('BRAIN'))

    pf.memorize('ST_SURFACE').set('PANEL', pf, 'PROP', pf.H_BRAIN).get('SETUP')

    pf.memorize('ST_AMBIENT').set('PANEL', pf, 'PROP', pf.H_AXES).get('SETUP')
end

%%% ¡prop!
SURF (metadata, item) is the brain surface to be plotted. ⑩
%%%% ¡settings!
'BrainSurface'
%%%% ¡default!
ImporterBrainSurfaceNV('FILE', BrainSurfacePF.getPropDefault('SURFFILE')).get('SURF')

%%% ¡prop!
H_BRAIN (evanescent, handle) is the handle for brain surface. ⑪
%%%% ¡calculate! ⑫
triangles = pf.get('SURF').get('TRIANGLES');
coordinates = pf.get('SURF').get('COORDINATES');
```

⑤ determines the axis setting through the container property `SettingsAxis`, which derives from `Settings`.

⑥ defines the default values by instantiating a default instance of `SettingsAxis`.

⑦ employs the property panel `SettingsAxisPP`, which is specialized for `SettingsAxis` and derives from `SettingsPP`.

⑧ contains the file from which the brain surface is plotted.

⑨ is executed only when the `SURFFILE` property is set. It updates the property `SURF` loading the data from the file. It the figure panel is already drawn, it refreshes the brain handle and redraws it.

⑩ contains the `BrainSurface` element.

⑪ is the evanescent handle for the brain surface. This is calcualted by ⑫.

```
h_brain = trisurf( ...
    triangles, ...
    coordinates(:, 1), ...
    coordinates(:, 2), ...
    coordinates(:, 3), ...
    'Parent', pf.memorize('H_AXES'), ...
    'Tag', 'H_BRAIN' ...
    );
xlabel(pf.get('H_AXES'), 'Sagittal')
ylabel(pf.get('H_AXES'), 'Axial')
zlabel(pf.get('H_AXES'), 'Coronal')
value = h_brain;

%%% ¡prop!
BRAIN (figure, logical) determines whether the brain surface is shown. (13)
%%%% ¡default!
true
%%%% ¡postset!
if pf.get('DRAWN')
    if pf.get('BRAIN')
        set(pf.get('H_BRAIN'), 'Visible', 'on')
    else % ~pf.get('BRAIN')
        set(pf.get('H_BRAIN'), 'Visible', 'off')
    end
end

%%% ¡prop!
ST_SURFACE (figure, item) determines the surface settings. (14)
%%%% ¡settings!
'SettingsSurface'
%%%% ¡gui!
pr = SettingsSurfacePP('EL', pf, 'PROP', BrainSurfacePF.ST_SURFACE, varargin
    {:}); (15)

%%% ¡prop!
ST_AMBIENT (figure, item) determines the ambient settings. (16)
%%%% ¡settings!
'SettingsAmbient'
%%%% ¡default!
SettingsAmbient('LIGHTING', 'gouraud', 'MATERIAL', 'dull', 'CAMLIGHT', '
    headlight (x2)', 'SHADING', 'none', 'COLORMAP', 'none') (17)
%%%% ¡gui!
pr = SettingsAmbientPP('EL', pf, 'PROP', BrainSurfacePF.ST_AMBIENT, varargin
    {:}); (18)
```

(13) determines whether the brain surface is shown.

(14) determines the brain surface settings throught the container property `SettingsSurface`, which derives from `Settings`.

(15) employs the property panel `SettingsSurfacePP`, which is specialized for `SettingsSurface` and derives from `SettingsPP`.

(16) determines the ambient lighting settings throught the container property `SettingsAmbient`, which is derived from `Settings`.

(17) defines the default values by instantiating a default instance of `SettingsAmbient`.

(18) employs the property panel `SettingsAmbientPP`, which is specialized for `SettingsAmbient` and derives from `SettingsPP`.

Code 4: **BrainSurfacePF element props update.** The `props_update` section of the generator code for `_BrainSurfacePF.gen.m` updates the properties of the `PanelFig` element. This defines the core properties of the property panel.

```
%% ¡props_update!
...
%%% ¡prop!
DRAW (query, logical) draws the figure brain surface. (1)
%%%% ¡calculate!
value = calculateValue@PanelFig(pf, PanelFig.DRAW, varargin{:}); (2)
```

(1) initializes the various graphical elements are drawn.

(2) calls the constructor of the parent. It returns `value = true` if the panel is drawn correctly. It gives a warning if the panel is not drawn correctly.

```
if value
    pf.memorize('H_AXES') ③

    pf.memorize('ST_AXIS').set('PANEL', pf, 'PROP', BrainSurfacePF.H_AXES).
     get('SETUP') ④

    pf.memorize('H_BRAIN') ⑤

    pf.memorize('ST_SURFACE').set('PANEL', pf, 'PROP', BrainSurfacePF.
     H_BRAIN).get('SETUP') ⑥

    pf.memorize('ST_AMBIENT').set('PANEL', pf, 'PROP', BrainSurfacePF.H_AXES
     ).get('SETUP') ⑦
end

%%% ¡prop!
DELETE (query, logical) resets the handles when the panel figure brain
     surface is deleted. ⑧
%%%% ¡calculate!
value = calculateValue@PanelFig(pf, PanelFig.DELETE, varargin{:}); % also
     warning
if value
    pf.set('H_AXES', Element.getNoValue())
    pf.set('H_BRAIN', Element.getNoValue())
end
```

③ ensures that the axes are memorized.

④ creates, memorizes, and sets up the property H_AXES.

⑤ memorizes the property H_BRAIN.

⑥ creates, memorizes, and sets up the property ST_SURFACE.

⑦ creates, memorizes, and sets up the property ST_AMBIENT.

⑧ deletes all evanescent hnadles when the figure containing the panel is deleted.

Code 5: **BrainSurfacePF element tests.** The `tests` section of the generator code for `_BrainSurfacePF.gen.m` determines how the unit tests are performed.

```
%% ¡tests!

%%% ¡excluded_props! ①
[BrainSurfacePF.PARENT BrainSurfacePF.H BrainSurfacePF.ST_POSITION
     BrainSurfacePF.ST_AXIS BrainSurfacePF.ST_SURFACE BrainSurfacePF.
     ST_AMBIENT]

%%% ¡warning_off!
true

%%% ¡test!
%%%% ¡name!
Remove Figures
%%%% ¡code!
warning('off', [BRAPH2.STR ':BrainSurfacePF'])
assert(length(findall(0, 'type', 'figure')) == 1) ②
delete(findall(0, 'type', 'figure')) ③
warning('on', [BRAPH2.STR ':BrainSurfacePF'])
```

① some properties need to be excluded from the tests, mainly because they are initialized by other proprties and therefore could give some spurious errors.

② throws an error if there remains a different number of figures than expected.

③ removes the figures remaining from the testing.

Addition of Toolbar Buttons

We will now see how to add the pushbuttons in the toolbar of the figure, opportunely altering the code so far implemented.

Code 6: **BrainSurfacePF element props update.** The `props_update` section of the generator code for `_BrainSurfacePF.gen.m` with the additions needed to have the toolbar pushbuttons. ← Code 4

```
%% ¡props_update!
...
%%% ¡prop!
H_TOOLS (evanescent, handlelist) is the list of panel-specific tools from the first. ①
%%%% ¡calculate!
toolbar = pf.memorize('H_TOOLBAR'); ②
if check_graphics(toolbar, 'uitoolbar') ③
    value = calculateValue@PanelFig(pf, PanelFig.H_TOOLS);

    tool_separator_1 = uipushtool(toolbar, 'Separator', 'on', 'Visible', 'off');

    % Brain
    tool_brain = uitoggletool(toolbar, ...
        'Tag', 'TOOL.Brain', ...
        'Separator', 'on', ...
        'State', pf.get('BRAIN'), ...
        'Tooltip', 'Show Brain', ...
        'CData', imread('icon_brain.png'), ...
        'OnCallback', {@cb_brain, true}, ...
        'OffCallback', {@cb_brain, false});

    % Axis
    tool_axis = uitoggletool(toolbar, ...
        'Tag', 'TOOL.Axis', ...
        'State', pf.get('ST_AXIS').get('AXIS'), ...
        'Tooltip', 'Show axis', ...
        'CData', imread('icon_axis.png'), ...
        'OnCallback', {@cb_axis, true}, ...
        'OffCallback', {@cb_axis, false});

    % Grid
    tool_grid = uitoggletool(toolbar, ...
        'Tag', 'TOOL.Grid', ...
        'State', pf.get('ST_AXIS').get('GRID'), ...
        'Tooltip', 'Show grid', ...
        'CData', imread('icon_grid.png'), ...
        'OnCallback', {@cb_grid, true}, ...
        'OffCallback', {@cb_grid, false});

    tool_separator_2 = uipushtool(toolbar, 'Separator', 'on', 'Visible', 'off');

    % View 3D
    tool_view3D = uitoggletool(toolbar, ...
        'Tag', 'TOOL.View3D', ...
        'Separator', 'on', ...
        'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_3D_AZEL), ...
        'Tooltip', BrainSurfacePF.VIEW_3D_CMD, ...
        'CData', imread('icon_view_3d.png'), ...
        'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_3D_AZEL});

    % View SL
    tool_viewSL = uitoggletool(toolbar, ...
        'Tag', 'TOOL.ViewSL', ...
```

① provides a list of evanescent handles to toolbar pushbuttons.

② retrieves the toolbar and ③ checks that it is actually drawn.

```
        'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_SL_AZEL), ...
        'Tooltip', BrainSurfacePF.VIEW_SL_CMD, ...
        'CData', imread('icon_view_sl.png'), ...
        'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_SL_AZEL});

    % View SR
    tool_viewSR = uitoggletool(toolbar, ...
        'Tag', 'TOOL.ViewSR', ...
        'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_SR_AZEL), ...
        'Tooltip', BrainSurfacePF.VIEW_SR_CMD, ...
        'CData', imread('icon_view_sr.png'), ...
        'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_SR_AZEL});

    % View AD
    tool_viewAD = uitoggletool(toolbar, ...
        'Tag', 'TOOL.ViewAD', ...
        'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_AD_AZEL), ...
        'Tooltip', BrainSurfacePF.VIEW_AD_CMD, ...
        'CData', imread('icon_view_ad.png'), ...
        'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_AD_AZEL});

    % View AV
    tool_viewAV = uitoggletool(toolbar, ...
        'Tag', 'TOOL.ViewAV', ...
        'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_AV_AZEL), ...
        'Tooltip', BrainSurfacePF.VIEW_AV_CMD, ...
        'CData', imread('icon_view_av.png'), ...
        'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_AV_AZEL});

    % View CA
    tool_viewCA = uitoggletool(toolbar, ...
        'Tag', 'TOOL.ViewCA', ...
        'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_CA_AZEL), ...
        'Tooltip', BrainSurfacePF.VIEW_CA_CMD, ...
        'CData', imread('icon_view_ca.png'), ...
        'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_CA_AZEL});

    % View CP
    tool_viewCP = uitoggletool(toolbar, ...
        'Tag', 'TOOL.ViewCP', ...
        'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_CP_AZEL), ...
        'Tooltip', BrainSurfacePF.VIEW_CP_CMD, ...
        'CData', imread('icon_view_cp.png'), ...
        'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_CP_AZEL});

    value = {value{:}, ... (4)
        tool_separator_1, ...
        tool_brain, tool_axis, tool_grid, ...
        tool_separator_2, ...
        tool_view3D, tool_viewSL, tool_viewSL, tool_viewSR, tool_viewAD,
     tool_viewAV, tool_viewCA, tool_viewCP ...
        };
else
    value = {};
end
%%%% ¡calculate_callbacks! (5)
function cb_brain(~, ~, brain) % (src, event)
    pf.set('BRAIN', brain)
end
function cb_axis(~, ~, axis) % (src, event)
    pf.get('ST_AXIS').set('AXIS', axis);
```

(4) reorders the pushbuttons.

(5) provides the callback functions for the pushbuttons.

```

    % triggers the update of ST_AXIS
    pf.set('ST_AXIS', pf.get('ST_AXIS'))
end
function cb_grid(~, ~, grid) % (src, event)
    pf.get('ST_AXIS').set('GRID', grid);

    % triggers the update of ST_AXIS
    pf.set('ST_AXIS', pf.get('ST_AXIS'))
end
function cb_view(~, ~, azel) % (src, event)
    pf.set('VIEW', azel)
end

%%% ¡prop!
DRAW (query, logical) draws the figure brain surface.
%%%% ¡calculate!
value = calculateValue@PanelFig(pf, PanelFig.DRAW, varargin{:});
if value
    pf.memorize('H_AXES')

    pf.set('VIEW', pf.get('VIEW')) ①

    pf.memorize('ST_AXIS').set('PANEL', pf, 'PROP', BrainSurfacePF.H_AXES).get('SETUP')
    pf.memorize('LISTENER_ST_AXIS'); ②

    pf.memorize('H_BRAIN')

    pf.set('BRAIN', pf.get('BRAIN')) ③

    pf.memorize('ST_SURFACE').set('PANEL', pf, 'PROP', BrainSurfacePF.H_BRAIN).get('SETUP')

    pf.memorize('ST_AMBIENT').set('PANEL', pf, 'PROP', BrainSurfacePF.H_AXES).get('SETUP')
end

%%% ¡prop!
DELETE (query, logical) resets the handles when the panel figure brain surface is deleted.
%%%% ¡calculate!
value = calculateValue@PanelFig(pf, PanelFig.DELETE, varargin{:});
if value
    pf.set('H_AXES', Element.getNoValue())
    pf.set('H_BRAIN', Element.getNoValue())

    pf.set('LISTENER_ST_AXIS', Element.getNoValue()) ④
end
```

① ensures that the `postset` code is executed by resetting `VIEW` to its current value. This is needed to update the toolbar pushbuttons when the figure panel is first drawn.

② memorizes also the listener to the changes in `ST_AXIS`. This is neede to ensure that the toolbar pushbuttons are synchronized with the content of `ST_AXIS`.

③ ensures that the `postset` code is executed by resetting `BRAIN` to its current value. This is needed to update the toolbar pushbuttons when the figure panel is first drawn.

④ deletes also the evanescent handle for the `LISTENER_ST_AXIS`.

Code 7: **BrainSurfacePF element new props with toolbar pushbuttons.** The `props` section of the generator code for `_BrainSurfacePF.gen.m` with the additions needed to have the toolbar pushbuttons for the brain surface. ← Code 3

```
%% ¡props!
...
%%% ¡prop!
```

```
VIEW (figure, rvector) sets the desired view as the line-of-sight azimuth
     and elevation angles.
%%%% ¡check_prop!
check = length(value) == 2;
%%%% ¡default!
BrainSurfacePF.VIEW_SL_AZEL
%%%% ¡postset!
if pf.get('DRAWN')
    view(pf.get('H_AXES'), pf.get('VIEW'))

    % reset the ambient lighting
    pf.memorize('ST_AMBIENT').set('PANEL', pf, 'PROP', pf.H_AXES).get('SETUP
     ')

    % update state of toggle tools ①
    toolbar = pf.get('H_TOOLBAR');
    if check_graphics(toolbar, 'uitoolbar')
        set(findobj(toolbar, 'Tag', 'TOOL.View3D'), 'State', isequal(pf.get(
     'VIEW'), BrainSurfacePF.VIEW_3D_AZEL))
        set(findobj(toolbar, 'Tag', 'TOOL.ViewSL'), 'State', isequal(pf.get(
     'VIEW'), BrainSurfacePF.VIEW_SL_AZEL))
        set(findobj(toolbar, 'Tag', 'TOOL.ViewSR'), 'State', isequal(pf.get(
     'VIEW'), BrainSurfacePF.VIEW_SR_AZEL))
        set(findobj(toolbar, 'Tag', 'TOOL.ViewAD'), 'State', isequal(pf.get(
     'VIEW'), BrainSurfacePF.VIEW_AD_AZEL))
        set(findobj(toolbar, 'Tag', 'TOOL.ViewAV'), 'State', isequal(pf.get(
     'VIEW'), BrainSurfacePF.VIEW_AV_AZEL))
        set(findobj(toolbar, 'Tag', 'TOOL.ViewCA'), 'State', isequal(pf.get(
     'VIEW'), BrainSurfacePF.VIEW_CA_AZEL))
        set(findobj(toolbar, 'Tag', 'TOOL.ViewCP'), 'State', isequal(pf.get(
     'VIEW'), BrainSurfacePF.VIEW_CP_AZEL))
    end
end
%%%% ¡gui!
pr = PanelPropRVectorView('EL', pf, 'PROP', BrainSurfacePF.VIEW, varargin
     {:});

%%% ¡prop!
ST_AXIS (figure, item) determines the axis settings.
%%%% ¡settings!
'SettingsAxis'
%%%% ¡default!
SettingsAxis('GRID', false, 'AXIS', false)
%%%% ¡postset! ②
if pf.get('DRAWN')
    toolbar = pf.get('H_TOOLBAR');
    if check_graphics(toolbar, 'uitoolbar')
        set(findobj(toolbar, 'Tag', 'TOOL.Grid'), 'State', pf.get('ST_AXIS')
     .get('GRID'))
        set(findobj(toolbar, 'Tag', 'TOOL.Axis'), 'State', pf.get('ST_AXIS')
     .get('AXIS'))
    end
end
%%%% ¡gui!
pr = SettingsAxisPP('EL', pf, 'PROP', BrainSurfacePF.ST_AXIS, varargin{:});

%%% ¡prop! ③
LISTENER_ST_AXIS (evanescent, handle) contains the listener to the axis
     settings to update the pushbuttons.
%%%% ¡calculate!
```

① ensures that toolbar pushbuttons are updated with the current view.

② ensures that the toolbar pushbuttons are updated whenever the ST_AXIS property is updated.

③ ensures that the toolbar pushbuttons are updated whenever the ST_AXIS property is updated.

```
value = listener(pf.get('ST_AXIS'), 'PropSet', @cb_listener_st_axis);
%%%% ¡calculate_callbacks!
function cb_listener_st_axis(~, ~)
    if pf.get('DRAWN')
        toolbar = pf.get('H_TOOLBAR');
        if check_graphics(toolbar, 'uitoolbar')
            set(findobj(toolbar, 'Tag', 'TOOL.Grid'), 'State', pf.get('ST_AXIS').get('GRID'))
            set(findobj(toolbar, 'Tag', 'TOOL.Axis'), 'State', pf.get('ST_AXIS').get('AXIS'))
        end
    end
end
...
%%% ¡prop!
BRAIN (figure, logical) determines whether the brain surface is shown.
%%%% ¡default!
true
%%%% ¡postset!
if pf.get('DRAWN')
    if pf.get('BRAIN')
        set(pf.get('H_BRAIN'), 'Visible', 'on')
    else % ~pf.get('BRAIN')
        set(pf.get('H_BRAIN'), 'Visible', 'off')
    end

    toolbar = pf.get('H_TOOLBAR'); ③
    if check_graphics(toolbar, 'uitoolbar')
        set(findobj(toolbar, 'Tag', 'TOOL.Brain'), 'State', pf.get('BRAIN'))
    end
end
...
```

③ ensures that the toolbar pushbuttons are updated whenever the `BRAIN` property is updated.

Code 8: **BrainSurfacePF element tests with toolbar pushbuttons.** The `tests` section of the generator code for `_BrainSurfacePF.gen.m` with the additions needed to have the tool- bar pushbuttons for the brain surface. ← Code 5

```
%% ¡tests!

%%% ¡excluded_props!
[BrainSurfacePF.PARENT BrainSurfacePF.H BrainSurfacePF.ST_POSITION BrainSurfacePF.ST_AXIS BrainSurfacePF.ST_SURFACE BrainSurfacePF.ST_AMBIENT BrainSurfacePF.LISTENER_ST_AXIS] ①
...
```

① excludes from testing also `LISTENER_ST_AXIS`.

Extension of Figure Panel (BrainAtlasPF)

We will now explore how to extend `BrainSurfacePF` to plot also brain regions. We will therefore implement `BrainAtlasPF`.

Code 9: **BrainAtlasPF element header.** The `header` section of the generator code for `_BrainAtlasPF.gen.m` provides the general information about the `BrainAtlasPF` element.

```
%% iheader!
BrainAtlasPF < BrainSurfacePF (pf, panel figure brain atlas) is a plot of a
    brain atlas.

%%% idescription!
BrainAtlasPF manages the plot of the brain regions symbols,
spheres, ids and labels. BrainAtlasPF utilizes the surface created
from PFBrainSurface to integrate the regions to a brain surface.

%%% iseealso!
BrainAtlas, BrainSurface
```

Code 10: **BrainAtlasPF spheres.** This code demonstrates how to add the spheres to the `BrainAtlasPF`.

```
%% iprops!

%%% iprop! ①
BA (metadata, item) is the brain atlas with the brain regions.
%%%% isettings!
'BrainAtlas'

%%% iprop! ②
H_SPHS (evanescent, handlelist) is the set of handles for the spheres.
%%%% icalculate! ③
L = pf.memorize('BA').get('BR_DICT').get('LENGTH');
h_sphs = cell(1, L);
for i = 1:1:L
    h_sphs{i} = surf([], [], [], ...
        'Parent', pf.memorize('H_AXES'), ...
        'Tag', ['H_SPHS{' int2str(i) '}'], ...
        'Visible', false ...
        );
end
value = h_sphs;

%%% iprop! ④
SPHS (figure, logical) determines whether the spheres are shown.
%%%% idefault!
true
%%%% ipostset!
if ~pf.get('SPHS') % false ⑤
    h_sphs = pf.get('H_SPHS');
    for i = 1:1:length(h_sphs)
        set(h_sphs{i}, 'Visible', false)
    end
else % true ⑥
    % triggers the update of SPH_DICT
```

① containes the brain atlas to be visualized.

② contains the evanescent handles for the spehres. ③ draws the spheres and creates the handles.

④ determines whether the shperes are shown. When it is set to `FALSE`, ⑤ sets all spheres already drawn to invisible. When it is set to `TRUE`, ⑥ triggers the update of the sphere dictionary containing the elements corresponding to each sphere.

```
    pf.set('SPH_DICT', pf.get('SPH_DICT'))
end

%%% ¡prop!
SPH_DICT (figure, idict) contains the spheres of the brain regions. (7)
%%%% ¡settings!
'SettingsSphere'
%%%% ¡postset!
if pf.get('SPHS') && ~isa(pf.getr('BA'), 'NoValue') (8)

    br_dict = pf.get('BA').get('BR_DICT');

    if pf.get('SPH_DICT').get('LENGTH') == 0 && br_dict.get('LENGTH') (9)
        for i = 1:1:br_dict.get('LENGTH')
            br = br_dict.get('IT', i);
            sphs{i} = SettingsSphere( ...
                'PANEL', pf, ...
                'PROP', BrainAtlasPF.H_SPHS, ...
                'I', i, ...
                'VISIBLE', true, ...
                'ID', br.get('ID'), ...
                'X', br.get('X'), ...
                'Y', br.get('Y'), ...
                'Z', br.get('Z'), ...
                'FACECOLOR', BRAPH2.COL, ...
                'FACEALPHA', 1 ...
                );
        end
        pf.get('SPH_DICT').set('IT_LIST', sphs)
    end

    for i = 1:1:br_dict.get('LENGTH') (10)
        pf.get('SPH_DICT').get('IT', i).get('SETUP')
    end

    % reset the ambient lighting
    pf.get('ST_AMBIENT').get('SETUP')
end
%%%% ¡gui! (11)
pr = PanelPropIDictTable('EL', pf, 'PROP', BrainAtlasPF.SPH_DICT, ...
    'COLS', [PanelPropIDictTable.SELECTOR SettingsSphere.VISIBLE
     SettingsSphere.X SettingsSphere.Y SettingsSphere.Z SettingsSphere.
     SPHERESIZE SettingsSphere.FACECOLOR SettingsSphere.FACEALPHA
     SettingsSphere.EDGECOLOR SettingsSphere.EDGEALPHA], ...
    varargin{:});

%% ¡props_update!

...

%%% ¡prop!
DRAW (query, logical) draws the figure brain atlas.
%%%% ¡calculate!
value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:});
      % also warning
if value
    pf.memorize('H_SPHS') (12)
    pf.set('SPHS', pf.get('SPHS')) (13)
```

(7) provides the dictionary with all sphere elements, which is only executed if (8) the brain atlas is set.

(9) creates the sphere elements if they do not already exist. Each sphere element is a `SettingsSphere` with all properties necessary to set the sphere.

(10) setups the sphere objects by calling the property `SETUP` on each of them.

(11) uses `PanelPropIDictTable` to provide a table where the sphere settings can be managed.

(12) memorizes the sphere handles.

(13) sets the sphere elements `SettingsSphere` by triggering the `postset` of `SPHS`.

```

    % reset the ambient lighting
    pf.get('ST_AMBIENT').get('SETUP')
end

%%% ¡prop!
DELETE (query, logical) resets the handles when the panel figure brain
     surface is deleted.
%%%% ¡calculate!
value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DELETE, varargin
     {:}); % also warning
if value
    pf.set('H_SPHS', Element.getNoValue()) (14)
end

%% ¡tests!

%%% ¡excluded_props!
[BrainAtlasPF.PARENT BrainAtlasPF.H BrainAtlasPF.ST_POSITION BrainAtlasPF.
     ST_AXIS BrainAtlasPF.ST_SURFACE BrainAtlasPF.ST_AMBIENT]

...
```

(14) deletes the sphere handles when the figure panel is deleted.

Code 11: **BrainAtlasPF symbols.** This code demonstrates how to add the symbols to the BrainAtlasPF. ← Code 10

```
%% ¡props!

...

%%% ¡prop!
H_SYMS (evanescent, handlelist) is the set of handles for the symbols.
%%%% ¡calculate!
L = pf.memorize('BA').get('BR_DICT').get('LENGTH');
h_syms = cell(1, L);
for i = 1:1:L
    h_syms{i} = plot3(0, 0, 0, ...
        'Parent', pf.get('H_AXES'), ...
        'Tag', ['H_SYMS{' int2str(i) '}'], ...
        'Visible', false ...
        );
end
value = h_syms;

%%% ¡prop!
SYMS (figure, logical) determines whether the symbols are shown.
%%%% ¡default!
false
%%%% ¡postset!
if ~pf.get('SYMS') % false
    h_syms = pf.get('H_SYMS');
    for i = 1:1:length(h_syms)
        set(h_syms{i}, 'Visible', false)
    end
else % true
    % triggers the update of SYM_DICT
    pf.set('SYM_DICT', pf.get('SYM_DICT'))
end

%%% ¡prop!
```

```
SYM_DICT (figure, idict) contains the symbols of the brain regions.
%%%% ¡settings!
'SettingsSymbol'
%%%% ¡postset!
if pf.get('SYMS') && ~isa(pf.getr('BA'), 'NoValue')

    br_dict = pf.get('BA').get('BR_DICT');

  if pf.get('SYM_DICT').get('LENGTH') == 0 && br_dict.get('LENGTH')
        for i = 1:1:br_dict.get('LENGTH')
            br = br_dict.get('IT', i);
            syms{i} = SettingsSymbol( ...
                'PANEL', pf, ...
                'PROP', BrainAtlasPF.H_SYMS, ...
                'I', i, ...
                'VISIBLE', true, ...
                'ID', br.get('ID'), ... % Callback('EL', br, 'TAG', 'ID'),
     ...
                'X', br.get('X'), ... % Callback('EL', br, 'TAG', 'X'), ...
                'Y', br.get('Y'), ... % Callback('EL', br, 'TAG', 'Y'), ...
                'Z', br.get('Z') ... % Callback('EL', br, 'TAG', 'Z') ...
                );
        end
        pf.get('SYM_DICT').set('IT_LIST', syms)
    end

    for i = 1:1:br_dict.get('LENGTH')
        pf.get('SYM_DICT').get('IT', i).get('SETUP')
    end
end
%%%% ¡gui!
pr = PanelPropIDictTable('EL', pf, 'PROP', BrainAtlasPF.SYM_DICT, ...
    'COLS', [PanelPropIDictTable.SELECTOR SettingsSymbol.VISIBLE
     SettingsSymbol.X SettingsSymbol.Y SettingsSymbol.Z SettingsSymbol.
     SYMBOL SettingsSymbol.SYMBOLSIZE SettingsSymbol.EDGECOLOR
     SettingsSymbol.FACECOLOR], ...
    varargin{:});

%% ¡props_update!

...

%%% ¡prop!
DRAW (query, logical) draws the figure brain atlas.
%%%% ¡calculate!
value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:});
      % also warning
if value
    pf.memorize('H_SPHS')
    pf.set('SPHS', pf.get('SPHS'))

    pf.memorize('H_SYMS')
    pf.set('SYMS', pf.get('SYMS'))

    % reset the ambient lighting
    pf.get('ST_AMBIENT').get('SETUP')
end

%%% ¡prop!
DELETE (query, logical) resets the handles when the panel figure brain
     surface is deleted.
```

```
%%%% ¡calculate!
value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DELETE, varargin
    {:}); % also warning
if value
    pf.set('H_SPHS', Element.getNoValue())
    pf.set('H_SYMS', Element.getNoValue())
end

...
```

Code 12: **BrainAtlasPF ids.** This code demonstrates how to add the ids to the BrainAtlasPF. ← Code 11

```
%% ¡props!

...

%%% ¡prop!
H_IDS (evanescent, handlelist) is the set of handles for the ids.
%%%% ¡calculate!
L = pf.memorize('BA').get('BR_DICT').get('LENGTH');
h_ids = cell(1, L);
for i = 1:1:L
    h_ids{i} = text(0, 0, 0, '', ...
        'Parent', pf.get('H_AXES'), ...
        'Tag', ['H_IDS{' int2str(i) '}'], ...
        'Visible', false ...
        );
end
value = h_ids;

%%% ¡prop!
IDS (figure, logical) determines whether the ids are shown.
%%%% ¡default!
false
%%%% ¡postset!
if ~pf.get('IDS') % false
    h_ids = pf.get('H_IDS');
    for i = 1:1:length(h_ids)
        set(h_ids{i}, 'Visible', false)
    end
else % true
    % triggers the update of ID_DICT
    pf.set('ID_DICT', pf.get('ID_DICT'))
end

%%% ¡prop!
ID_DICT (figure, idict) contains the ids of the brain regions.
%%%% ¡settings!
'SettingsText'
%%%% ¡postset!
if pf.get('IDS') && ~isa(pf.getr('BA'), 'NoValue')

    br_dict = pf.get('BA').get('BR_DICT');

    if pf.get('ID_DICT').get('LENGTH') == 0 && br_dict.get('LENGTH')
        for i = 1:1:br_dict.get('LENGTH')
            br = br_dict.get('IT', i);
            ids{i} = SettingsText( ...
                'PANEL', pf, ...
                'PROP', BrainAtlasPF.H_IDS, ...
                'I', i, ...
                'VISIBLE', true, ...
                'ID', br.get('ID'), ... % Callback('EL', br, 'TAG', 'ID'),
     ...
                'X', br.get('X'), ... % Callback('EL', br, 'TAG', 'X'), ...
                'Y', br.get('Y'), ... % Callback('EL', br, 'TAG', 'Y'), ...
                'Z', br.get('Z'), ... % Callback('EL', br, 'TAG', 'Z'), ...
                'TXT', br.get('ID') ... % Callback('EL', br, 'TAG', 'ID')
     ...
                );
        end
        pf.get('ID_DICT').set('IT_LIST', ids)
    end

    for i = 1:1:br_dict.get('LENGTH')
        pf.get('ID_DICT').get('IT', i).get('SETUP')
    end
end
%%%% ¡gui!
pr = PanelPropIDictTable('EL', pf, 'PROP', BrainAtlasPF.ID_DICT, ...
    'COLS', [PanelPropIDictTable.SELECTOR SettingsText.VISIBLE SettingsText.
     X SettingsText.Y SettingsText.Z SettingsText.ROTATION SettingsText.TXT
     SettingsText.FONTNAME SettingsText.FONTSIZE SettingsText.FONTCOLOR
     SettingsText.INTERPRETER], ...
    varargin{:});

%% ¡props_update!

...

%%% ¡prop!
DRAW (query, logical) draws the figure brain atlas.
%%%% ¡calculate!
value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:});
      % also warning
if value
    pf.memorize('H_SPHS')
    pf.set('SPHS', pf.get('SPHS'))

    pf.memorize('H_SYMS')
    pf.set('SYMS', pf.get('SYMS'))

    pf.memorize('H_IDS')
    pf.set('SPHS', pf.get('SPHS'))

    % reset the ambient lighting
    pf.get('ST_AMBIENT').get('SETUP')
end

%%% ¡prop!
DELETE (query, logical) resets the handles when the panel figure brain
     surface is deleted.
%%%% ¡calculate!
value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DELETE, varargin
     {:}); % also warning
if value
    pf.set('H_SPHS', Element.getNoValue())
    pf.set('H_SYMS', Element.getNoValue())
    pf.set('H_IDS', Element.getNoValue())
end

```

```
...
```

Code 13: **BrainAtlasPF labels.** This code demonstrates how to add the labels to the BrainAtlasPF. ← Code 12

```
%% ¡props!

...

%%% ¡prop!
H_LABS (evanescent, handlelist) is the set of handles for the labels.
%%%% ¡calculate!
L = pf.memorize('BA').get('BR_DICT').get('LENGTH');
h_labs = cell(1, L);
for i = 1:1:L
    h_labs{i} = text(0, 0, 0, '', ...
        'Parent', pf.get('H_AXES'), ...
        'Tag', ['H_LABS{' int2str(i) '}'], ...
        'Visible', false ...
        );
end
value = h_labs;

%%% ¡prop!
LABS (figure, logical) determines whether the labels are shown.
%%%% ¡default!
false
%%%% ¡postset!
if ~pf.get('LABS') % false
    h_labs = pf.get('H_LABS');
    for i = 1:1:length(h_labs)
        set(h_labs{i}, 'Visible', false)
    end
else % true
  % triggers the update of LAB_DICT
  pf.set('LAB_DICT', pf.get('LAB_DICT'))
end

%%% ¡prop!
LAB_DICT (figure, idict) contains the labels of the brain regions.
%%%% ¡settings!
'SettingsText'
%%%% ¡postset!
if pf.get('LABS') && ~isa(pf.getr('BA'), 'NoValue')

    br_dict = pf.get('BA').get('BR_DICT');

    if pf.get('LAB_DICT').get('LENGTH') == 0 && br_dict.get('LENGTH')
        for i = 1:1:br_dict.get('LENGTH')
            br = br_dict.get('IT', i);
            labs{i} = SettingsText( ...
                'PANEL', pf, ...
                'PROP', BrainAtlasPF.H_LABS, ...
                'I', i, ...
                'VISIBLE', true, ...
                'ID', br.get('ID'), ... % Callback('EL', br, 'TAG', 'ID'),
     ...
                'X', br.get('X'), ... % Callback('EL', br, 'TAG', 'X'), ...
                'Y', br.get('Y'), ... % Callback('EL', br, 'TAG', 'Y'), ...
                'Z', br.get('Z'), ... % Callback('EL', br, 'TAG', 'Z'), ...
                'TXT', br.get('LABEL') ... % Callback('EL', br, 'TAG', 'LABEL') ...
                );
        end
        pf.get('LAB_DICT').set('IT_LIST', labs)
    end

    for i = 1:1:br_dict.get('LENGTH')
        pf.get('LAB_DICT').get('IT', i).get('SETUP')
    end
end
%%%% ¡gui!
pr = PanelPropIDictTable('EL', pf, 'PROP', BrainAtlasPF.LAB_DICT, ...
    'COLS', [PanelPropIDictTable.SELECTOR SettingsText.VISIBLE SettingsText.X SettingsText.Y SettingsText.Z SettingsText.ROTATION SettingsText.TXT SettingsText.FONTNAME SettingsText.FONTSIZE SettingsText.FONTCOLOR SettingsText.INTERPRETER], ...
    varargin{:});

%% ¡props_update!

...

%%% ¡prop!
DRAW (query, logical) draws the figure brain atlas.
%%%% ¡calculate!
value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:}); % also warning
if value
    pf.memorize('H_SPHS')
    pf.set('SPHS', pf.get('SPHS'))

    pf.memorize('H_SYMS')
    pf.set('SYMS', pf.get('SYMS'))

    pf.memorize('H_IDS')
    pf.set('SPHS', pf.get('SPHS'))

    pf.memorize('H_LABS')
    pf.set('LABS', pf.get('LABS'))

    % reset the ambient lighting
    pf.get('ST_AMBIENT').get('SETUP')
end

%%% ¡prop!
DELETE (query, logical) resets the handles when the panel figure brain surface is deleted.
%%%% ¡calculate!
value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DELETE, varargin{:}); % also warning
if value
    pf.set('H_SPHS', Element.getNoValue())
    pf.set('H_SYMS', Element.getNoValue())
    pf.set('H_IDS', Element.getNoValue())
    pf.set('H_LABS', Element.getNoValue())
end

...
```

Extension of Toolbar Buttons

We will now see how to add toolbar pushbuttons to the previous code.

Code 14: **BrainAtlasPF with toolbar.** This code demonstrates how to add the toolbar pushbuttons to the `BrainAtlasPF`. ← Code 13

```
%% ¡props!

...

%%% ¡prop!
SPHS (figure, logical) determines whether the spheres are shown.
%%%% ¡default!
true
%%%% ¡postset!
if ~pf.get('SPHS') % false
    h_sphs = pf.get('H_SPHS');
    for i = 1:1:length(h_sphs)
        set(h_sphs{i}, 'Visible', false)
    end
else % true
    % triggers the update of SPH_DICT
    pf.set('SPH_DICT', pf.get('SPH_DICT'))
end

% update state of toggle tool
toolbar = pf.get('H_TOOLBAR');
if check_graphics(toolbar, 'uitoolbar')
    set(findobj(toolbar, 'Tag', 'TOOL.Sphs'), 'State', pf.get('SPHS'))
end

...

%%% ¡prop!
SYMS (figure, logical) determines whether the symbols are shown.
%%%% ¡default!
false
%%%% ¡postset!
if ~pf.get('SYMS') % false
    h_syms = pf.get('H_SYMS');
    for i = 1:1:length(h_syms)
        set(h_syms{i}, 'Visible', false)
    end
else % true
    % triggers the update of SYM_DICT
    pf.set('SYM_DICT', pf.get('SYM_DICT'))
end

% update state of toggle tool
toolbar = pf.get('H_TOOLBAR');
if check_graphics(toolbar, 'uitoolbar')
    set(findobj(toolbar, 'Tag', 'TOOL.Syms'), 'State', pf.get('SYMS'))
end

...

%%% ¡prop!
IDS (figure, logical) determines whether the ids are shown.
%%%% ¡default!
```

```
false
%%%% ¡postset!
if ~pf.get('IDS') % false
    h_ids = pf.get('H_IDS');
    for i = 1:1:length(h_ids)
        set(h_ids{i}, 'Visible', false)
    end
else % true
    % triggers the update of ID_DICT
    pf.set('ID_DICT', pf.get('ID_DICT'))
end

% update state of toggle tool
toolbar = pf.get('H_TOOLBAR');
if check_graphics(toolbar, 'uitoolbar')
    set(findobj(toolbar, 'Tag', 'TOOL.Ids'), 'State', pf.get('IDS'))
end

...

%%% ¡prop!
LABS (figure, logical) determines whether the labels are shown.
%%%% ¡default!
false
%%%% ¡postset!
if ~pf.get('LABS') % false
    h_labs = pf.get('H_LABS');
    for i = 1:1:length(h_labs)
        set(h_labs{i}, 'Visible', false)
    end
else % true
  % triggers the update of LAB_DICT
  pf.set('LAB_DICT', pf.get('LAB_DICT'))
end

% update state of toggle tool
toolbar = pf.get('H_TOOLBAR');
if check_graphics(toolbar, 'uitoolbar')
    set(findobj(toolbar, 'Tag', 'TOOL.Labs'), 'State', pf.get('LABS'))
end

...

%% ¡props_update!

...

%%% ¡prop!
DRAW (query, logical) draws the figure brain atlas.
%%%% ¡calculate!
value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:});
      % also warning
if value
    pf.memorize('H_SPHS')
    pf.set('SPHS', pf.get('SPHS')) % sets also   SPH_DICT

    pf.memorize('H_SYMS')
    pf.set('SYMS', pf.get('SYMS')) % sets also   SYM_DICT

    pf.memorize('H_IDS')
    pf.set('SPHS', pf.get('SPHS')) % sets also   ID_DICT

    pf.memorize('H_LABS')
    pf.set('LABS', pf.get('LABS')) % sets also   LAB_DICT

    % reset the ambient lighting
    pf.get('ST_AMBIENT').get('SETUP')
end
%%%% ¡calculate_callbacks!
function cb_sphs(~, ~, sphs) % (src, event)
    pf.set('SPHS', sphs)
end
function cb_syms(~, ~, syms) % (src, event)
    pf.set('SYMS', syms)
end
function cb_ids(~, ~, ids) % (src, event)
    pf.set('IDS', ids)
end
function cb_labs(~, ~, labs) % (src, event)
    pf.set('LABS', labs)
end

...

%%% ¡prop!
H_TOOLS (evanescent, handlelist) is the list of panel-specific tools from
     the first.
%%%% ¡calculate!
toolbar = pf.memorize(PanelFig.H_TOOLBAR);
if check_graphics(toolbar, 'uitoolbar')
  value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.H_TOOLS);

    tool_separator_1 = uipushtool(toolbar, 'Separator', 'on', 'Visible', '
     off');

    % Spheres
    tool_sphs = uitoggletool(toolbar, ...
        'Tag', 'TOOL.Sphs', ...
        'Separator', 'on', ...
        'State', pf.get('SPHS'), ...
        'Tooltip', 'Show Spheres', ...
        'CData', imread('icon_sphere.png'), ...
        'OnCallback', {@cb_sphs, true}, ...
        'OffCallback', {@cb_sphs, false});

    % Symbols
    tool_syms = uitoggletool(toolbar, ...
        'Tag', 'TOOL.Syms', ...
        'Separator', 'on', ...
        'State', pf.get('SYMS'), ...
        'Tooltip', 'Show Symbols', ...
        'CData', imread('icon_symbol.png'), ...
        'OnCallback', {@cb_syms, true}, ...
        'OffCallback', {@cb_syms, false});

    % IDs
    tool_ids = uitoggletool(toolbar, ...
        'Tag', 'TOOL.Ids', ...
        'Separator', 'on', ...
        'State', pf.get('IDS'), ...
        'Tooltip', 'Show IDs', ...
        'CData', imread('icon_id.png'), ...
        'OnCallback', {@cb_ids, true}, ...
        'OffCallback', {@cb_ids, false});

    % Labels
    tool_labs = uitoggletool(toolbar, ...
        'Tag', 'TOOL.Labs', ...
        'Separator', 'on', ...
        'State', pf.get('LABS'), ...
        'Tooltip', 'Show Labels', ...
        'CData', imread('icon_label.png'), ...
        'OnCallback', {@cb_labs, true}, ...
        'OffCallback', {@cb_labs, false});

    value = {value{:}, ...
        tool_separator_1, ...
        tool_sphs, tool_syms, tool_ids, tool_labs ...
        };
else
  value = {};
end

...
```