

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

<i>Implementation of Figure Panel (BrainSurfacePF)</i>	2
<i>Addition of Toolbar Buttons</i>	6
<i>Extension of Figure Panel (BrainAtlasPF)</i>	12
<i>Extension of Toolbar Buttons</i>	20

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.

```

1 %% iheader!
2 BrainSurfacePF < PanelFig (pf, panel figure brain surface) is a plot of a
   brain surface. ①
3
4 %% idescription!
5 BrainSurfacePF manages the plot of the brain surface chosen by the user.
6 A collection of brain surfaces in NV format can be found in the folder
7 ./braph2/brainsurfs/.
8 This class provides the common methods needed to manage the plot of
9 the surface. In particular, the user can change lighting, material,
10 camlight, shading, colormap, facecolor, brain color, face color,
11 edge color, and background color.
12
13 %% isealso!
14 BrainSurface

```

① The element BrainSurfacePF is defined as a subclass of PanelFig. The moniker will be pf.

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

```

1 %% iconstants!
2
3 % fixed 3d view
4 VIEW_3D = 1 % 3D view numeric code
5 VIEW_3D_CMD = '3D' % 3D view name
6 VIEW_3D_AZEL = [-37.5 30] % 3D view azimuthal and polar angles
7
8 % sagittal left view
9 VIEW_SL = 2 % sagittal left view numeric code
10 VIEW_SL_CMD = 'Sagittal left' % sagittal left view name
11 VIEW_SL_AZEL = [-90 0] % sagittal left view azimuthal and polar angles
12
13 % sagittal right view
14 VIEW_SR = 3 % sagittal right view numeric code
15 VIEW_SR_CMD = 'Sagittal right' % sagittal right view name
16 VIEW_SR_AZEL = [90 0] % sagittal right view azimuthal and polar angles
17
18 % axial dorsal view
19 VIEW_AD = 4 % axial dorsal view numeric code
20 VIEW_AD_CMD = 'Axial dorsal' % axial dorsal view name
21 VIEW_AD_AZEL = [0 90] % axial dorsal view azimuthal and polar angles
22
23 % axial ventral view
24 VIEW_AV = 5 % axial ventral view numeric code
25 VIEW_AV_CMD = 'Axial ventral' % axial ventral view name
26 VIEW_AV_AZEL = [0 -90] % axial ventral view azimuthal and polar angles
27

```

```

28 % coronal anterior view
29 VIEW_CA = 6 % coronal anterior view numeric code
30 VIEW_CA_CMD = 'Coronal anterior' % coronal anterior view name
31 VIEW_CA_AZEL = [180 0] % coronal anterior view azimuthal and polar angles
32
33 % coronal posterior view
34 VIEW_CP = 7 % coronal posterior view numeric code
35 VIEW_CP_CMD = 'Coronal posterior' % coronal posterior view name
36 VIEW_CP_AZEL = [0 0] % coronal posterior view azimuthal and polar angles
37
38 VIEW_CMD = { ... % vector of view names
39     BrainSurfacePF.VIEW_3D_CMD ...
40     BrainSurfacePF.VIEW_SL_CMD ...
41     BrainSurfacePF.VIEW_SR_CMD ...
42     BrainSurfacePF.VIEW_AD_CMD ...
43     BrainSurfacePF.VIEW_AV_CMD ...
44     BrainSurfacePF.VIEW_CA_CMD ...
45     BrainSurfacePF.VIEW_CP_CMD ...
46 }
47
48 VIEW_AZEL = { ... % vector of view azimuthal and polar angle
49     BrainSurfacePF.VIEW_3D_AZEL ...
50     BrainSurfacePF.VIEW_SL_AZEL ...
51     BrainSurfacePF.VIEW_SR_AZEL ...
52     BrainSurfacePF.VIEW_AD_AZEL ...
53     BrainSurfacePF.VIEW_AV_AZEL ...
54     BrainSurfacePF.VIEW_CA_AZEL ...
55     BrainSurfacePF.VIEW_CP_AZEL ...
56 }

```

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.

```

1 %% iprops!
2
3 %% iprop!
4 H_AXES (evanescent, handle) is the handle for the axes. ①
5 %% icalculate!
6 h_axes = uiaxes( ...
7     'Parent', pf.memorize('H'), ... ②
8     'Tag', 'H_AXES', ...
9     'Units', 'normalized', ...
10    'OuterPosition', [0 0 1 1] ...
11 );
12 h_axes.Toolbar.Visible = 'off';
13 h_axes.Interactions = [];
14 value = h_axes;
15
16 %% iprop!
17 VIEW (figure, rvector) sets the desired view as the line-of-sight azimuth
    and elevation angles. ③
18 %% icheck_prop!
19 check = length(value) == 2;
20 %% idefault!
21 BrainSurfacePF.VIEW_SL_AZEL
22 %% ipostset! ④
23 if pf.get('DRAWN')
24     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.

```

25
26 % reset the ambient lighting
27 pf.memorize('ST_AMBIENT').set('PANEL', pf, 'PROP', pf.H_AXES).get('SETUP
    ')
28 end
29 %%% igui!
30 pr = PanelPropRVectorView('EL', pf, 'PROP', BrainSurfacePF.VIEW, varargin
    {:});
31
32 %%% iprop!
33 ST_AXIS (figure, item) determines the axis settings. (5)
34 %%% isettings!
35 'SettingsAxis'
36 %%% idefault!
37 SettingsAxis('GRID', false, 'AXIS', false) (6)
38 %%% igui! (7)
39 pr = SettingsAxisPP('EL', pf, 'PROP', BrainSurfacePF.ST_AXIS, varargin{:});
40
41 %%% iprop!
42 SURFFILE (figure, option) is the name of the file of the brain surface to be
    plotted. (8)
43 %%% isettings!
44 {dir([fileparts(which('braph2')) filesep() 'brainsurfs' filesep() '*.nv')].
    name}
45 %%% idefault!
46 'human_ICBM152.nv'
47 %%% ipostset! (9)
48 bs = ImporterBrainSurfaceNV('FILE', pf.get('SURFFILE')).get('SURF');
49 pf.set('SURF', bs)
50
51 if pf.get('DRAWN')
52     delete(pf.get('H_BRAIN'))
53     pf.set('H_BRAIN', Element.getNoValue())
54
55     pf.memorize('H_BRAIN')
56
57     pf.set('BRAIN', pf.get('BRAIN'))
58
59     pf.memorize('ST_SURFACE').set('PANEL', pf, 'PROP', pf.H_BRAIN).get('
        SETUP')
60
61     pf.memorize('ST_AMBIENT').set('PANEL', pf, 'PROP', pf.H_AXES).get('SETUP
        ')
62 end
63
64 %%% iprop!
65 SURF (metadata, item) is the brain surface to be plotted. (10)
66 %%% isettings!
67 'BrainSurface'
68 %%% idefault!
69 ImporterBrainSurfaceNV('FILE', BrainSurfacePF.getPropDefault('SURFFILE')).
    get('SURF')
70
71 %%% iprop!
72 H_BRAIN (evanescent, handle) is the handle for brain surface. (11)
73 %%% icalculate! (12)
74 triangles = pf.get('SURF').get('TRIANGLES');
75 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. If 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 calculated by (12).

```

76 h_brain = trisurf( ...
77     triangles, ...
78     coordinates(:, 1), ...
79     coordinates(:, 2), ...
80     coordinates(:, 3), ...
81     'Parent', pf.memorize('H_AXES'), ...
82     'Tag', 'H_BRAIN' ...
83 );
84 xlabel(pf.get('H_AXES'), 'Sagittal')
85 ylabel(pf.get('H_AXES'), 'Axial')
86 zlabel(pf.get('H_AXES'), 'Coronal')
87 value = h_brain;
88
89 %%% iprop!
90 BRAIN (figure, logical) determines whether the brain surface is shown. (13)
91 %%% idefault!
92 true
93 %%% ipostset!
94 if pf.get('DRAWN')
95     if pf.get('BRAIN')
96         set(pf.get('H_BRAIN'), 'Visible', 'on')
97     else ~pf.get('BRAIN')
98         set(pf.get('H_BRAIN'), 'Visible', 'off')
99     end
100 end
101
102 %%% iprop!
103 ST_SURFACE (figure, item) determines the surface settings. (14)
104 %%% isettings!
105 'SettingsSurface'
106 %%% igui!
107 pr = SettingsSurfacePP('EL', pf, 'PROP', BrainSurfacePF.ST_SURFACE, varargin
    {:}); (15)
108
109 %%% iprop!
110 ST_ AMBIENT (figure, item) determines the ambient settings. (16)
111 %%% isettings!
112 'SettingsAmbient'
113 %%% idefault!
114 SettingsAmbient('LIGHTING', 'gouraud', 'MATERIAL', 'dull', 'CAMLIGHT', '
    headlight (x2)', 'SHADING', 'none', 'COLORMAP', 'none') (17)
115 %%% igui!
116 pr = SettingsAmbientPP('EL', pf, 'PROP', BrainSurfacePF.ST_ AMBIENT, varargin
    {:}); (18)

```

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.

```

1 %%% iprops_update!
2 ...
3 %%% iprop!
4 DRAW (query, logical) draws the figure brain surface. (1)
5 %%% icalculate!
6 value = calculateValue@PanelFig(pf, PanelFig.DRAW, varargin{:}); (2)

```

(13) determines whether the brain surface is shown.

(14) determines the brain surface settings through 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 through 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.

(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.

```

7 if value
8     pf.memorize('H_AXES') (3)
9
10    pf.memorize('ST_AXIS').set('PANEL', pf, 'PROP', BrainSurfacePF.H_AXES).
        get('SETUP') (4)
11
12    pf.memorize('H_BRAIN') (5)
13
14    pf.memorize('ST_SURFACE').set('PANEL', pf, 'PROP', BrainSurfacePF.
        H_BRAIN).get('SETUP') (6)
15
16    pf.memorize('ST_AMBIENT').set('PANEL', pf, 'PROP', BrainSurfacePF.H_AXES
        ).get('SETUP') (7)
17 end
18
19 %% iprop!
20 DELETE (query, logical) resets the handles when the panel figure brain
        surface is deleted. (8)
21 %%% calculate!
22 value = calculateValue@PanelFig(pf, PanelFig.DELETE, varargin{:}); % also
        warning
23 if value
24     pf.set('H_AXES', Element.getNoValue())
25     pf.set('H_BRAIN', Element.getNoValue())
26 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 handles 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.

```

1 %%% itests!
2
3 %%% iexcluded_props! (1)
4 [BrainSurfacePF.PARENT BrainSurfacePF.H BrainSurfacePF.ST_POSITION
    BrainSurfacePF.ST_AXIS BrainSurfacePF.ST_SURFACE BrainSurfacePF.
    ST_AMBIENT]
5
6 %%% iwarning_off!
7 true
8
9 %%% itest!
10 %%% iname!
11 Remove Figures
12 %%% icode!
13 warning('off', [BRAPH2.STR ':BrainSurfacePF'])
14 assert(length(findall(0, 'type', 'figure')) == 1) (2)
15 delete(findall(0, 'type', 'figure')) (3)
16 warning('on', [BRAPH2.STR ':BrainSurfacePF'])

```

(1) some properties need to be excluded from the tests, mainly because they are initialized by other properties 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](#)

```

1 %% iprops_update!
2 ...
3 %%% iprop!
4 H_TOOLS (evanescent, handlelist) is the list of panel-specific tools from
   the first. ①
5 %%% icalculate!
6 toolbar = pf.memorize('H_TOOLBAR'); ②
7 if check_graphics(toolbar, 'uitoolbar') ③
8     value = calculateValue@PanelFig(pf, PanelFig.H_TOOLS);
9
10    tool_separator_1 = uipushtool(toolbar, 'Separator', 'on', 'Visible', '
      off');
11
12 % Brain
13 tool_brain = uitoggletool(toolbar, ...
14     'Tag', 'TOOL.Brain', ...
15     'Separator', 'on', ...
16     'State', pf.get('BRAIN'), ...
17     'Tooltip', 'Show Brain', ...
18     'CData', imread('icon_brain.png'), ...
19     'OnCallback', {@cb_brain, true}, ...
20     'OffCallback', {@cb_brain, false});
21
22 % Axis
23 tool_axis = uitoggletool(toolbar, ...
24     'Tag', 'TOOL.Axis', ...
25     'State', pf.get('ST_AXIS').get('AXIS'), ...
26     'Tooltip', 'Show axis', ...
27     'CData', imread('icon_axis.png'), ...
28     'OnCallback', {@cb_axis, true}, ...
29     'OffCallback', {@cb_axis, false});
30
31 % Grid
32 tool_grid = uitoggletool(toolbar, ...
33     'Tag', 'TOOL.Grid', ...
34     'State', pf.get('ST_AXIS').get('GRID'), ...
35     'Tooltip', 'Show grid', ...
36     'CData', imread('icon_grid.png'), ...
37     'OnCallback', {@cb_grid, true}, ...
38     'OffCallback', {@cb_grid, false});
39
40    tool_separator_2 = uipushtool(toolbar, 'Separator', 'on', 'Visible', '
      off');
41
42 % View 3D
43 tool_view3D = uitoggletool(toolbar, ...
44     'Tag', 'TOOL.View3D', ...
45     'Separator', 'on', ...
46     'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_3D_AZEL), ...
47     'Tooltip', BrainSurfacePF.VIEW_3D_CMD, ...
48     'CData', imread('icon_view_3d.png'), ...
49     'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_3D_AZEL});
50
51 % View SL
52 tool_viewSL = uitoggletool(toolbar, ...
53     '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.

```

54     'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_SL_AZEL), ...
55     'Tooltip', BrainSurfacePF.VIEW_SL_CMD, ...
56     'CData', imread('icon_view_sl.png'), ...
57     'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_SL_AZEL});
58
59 % View SR
60 tool_viewSR = uitoggletool(toolbar, ...
61     'Tag', 'TOOL.ViewSR', ...
62     'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_SR_AZEL), ...
63     'Tooltip', BrainSurfacePF.VIEW_SR_CMD, ...
64     'CData', imread('icon_view_sr.png'), ...
65     'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_SR_AZEL});
66
67 % View AD
68 tool_viewAD = uitoggletool(toolbar, ...
69     'Tag', 'TOOL.ViewAD', ...
70     'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_AD_AZEL), ...
71     'Tooltip', BrainSurfacePF.VIEW_AD_CMD, ...
72     'CData', imread('icon_view_ad.png'), ...
73     'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_AD_AZEL});
74
75 % View AV
76 tool_viewAV = uitoggletool(toolbar, ...
77     'Tag', 'TOOL.ViewAV', ...
78     'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_AV_AZEL), ...
79     'Tooltip', BrainSurfacePF.VIEW_AV_CMD, ...
80     'CData', imread('icon_view_av.png'), ...
81     'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_AV_AZEL});
82
83 % View CA
84 tool_viewCA = uitoggletool(toolbar, ...
85     'Tag', 'TOOL.ViewCA', ...
86     'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_CA_AZEL), ...
87     'Tooltip', BrainSurfacePF.VIEW_CA_CMD, ...
88     'CData', imread('icon_view_ca.png'), ...
89     'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_CA_AZEL});
90
91 % View CP
92 tool_viewCP = uitoggletool(toolbar, ...
93     'Tag', 'TOOL.ViewCP', ...
94     'State', isequal(pf.get('VIEW'), BrainSurfacePF.VIEW_CP_AZEL), ...
95     'Tooltip', BrainSurfacePF.VIEW_CP_CMD, ...
96     'CData', imread('icon_view_cp.png'), ...
97     'ClickedCallback', {@cb_view, BrainSurfacePF.VIEW_CP_AZEL});
98
99 value = {value{:}, ... ④
100     tool_separator_1, ...
101     tool_brain, tool_axis, tool_grid, ...
102     tool_separator_2, ...
103     tool_view3D, tool_viewSL, tool_viewSL, tool_viewSR, tool_viewAD,
104     tool_viewAV, tool_viewCA, tool_viewCP ...
105 };
106 else
107     value = {};
108 end
109 %%% icalculate_callbacks! ⑤
110 function cb_brain(~, ~, brain) % (src, event)
111     pf.set('BRAIN', brain)
112 end
113 function cb_axis(~, ~, axis) % (src, event)
114     pf.set('ST_AXIS').set('AXIS', axis);

```

④ reorders the pushbuttons.

⑤ provides the callback functions for the pushbuttons.


```

114
115 % triggers the update of ST_AXIS
116 pf.set('ST_AXIS', pf.get('ST_AXIS'))
117 end
118 function cb_grid(~, ~, grid) % (src, event)
119     pf.get('ST_AXIS').set('GRID', grid);
120
121 % triggers the update of ST_AXIS
122 pf.set('ST_AXIS', pf.get('ST_AXIS'))
123 end
124 function cb_view(~, ~, azel) % (src, event)
125     pf.set('VIEW', azel)
126 end
127
128 %%% iprop!
129 DRAW (query, logical) draws the figure brain surface.
130 %%% icalculate!
131 value = calculateValue@PanelFig(pf, PanelFig.DRAW, varargin{:});
132 if value
133     pf.memorize('H_AXES')
134
135     pf.set('VIEW', pf.get('VIEW')) ①
136
137     pf.memorize('ST_AXIS').set('PANEL', pf, 'PROP', BrainSurfacePF.H_AXES).
138         get('SETUP')
139     pf.memorize('LISTENER_ST_AXIS'); ②
140
141     pf.memorize('H_BRAIN')
142
143     pf.set('BRAIN', pf.get('BRAIN')) ③
144
145     pf.memorize('ST_SURFACE').set('PANEL', pf, 'PROP', BrainSurfacePF.
146         H_BRAIN).get('SETUP')
147
148     pf.memorize('ST_ AMBIENT').set('PANEL', pf, 'PROP', BrainSurfacePF.H_AXES
149         ).get('SETUP')
150
151 end
152
153 %%% iprop!
154 DELETE (query, logical) resets the handles when the panel figure brain
155     surface is deleted.
156 %%% icalculate!
157 value = calculateValue@PanelFig(pf, PanelFig.DELETE, varargin{:});
158 if value
159     pf.set('H_AXES', Element.getNoValue())
160     pf.set('H_BRAIN', Element.getNoValue())
161
162     pf.set('LISTENER_ST_AXIS', Element.getNoValue()) ④
163 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 needed 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](#)

```

1 %%% iprops!
2 ...
3 %%% iprop!

```

```

4 VIEW (figure, rvector) sets the desired view as the line-of-sight azimuth
  and elevation angles.
5 %%% icheck_prop!
6 check = length(value) == 2;
7 %%% idefault!
8 BrainSurfacePF.VIEW_SL_AZEL
9 %%% ipostset!
10 if pf.get('DRAWN')
11     view(pf.get('H_AXES'), pf.get('VIEW'))
12
13     % reset the ambient lighting
14     pf.memorize('ST_AMBIENT').set('PANEL', pf, 'PROP', pf.H_AXES).get('SETUP
      ')
15
16     % update state of toggle tools ①
17     toolbar = pf.get('H_TOOLBAR');
18     if check_graphics(toolbar, 'uitoolbar')
19         set(findobj(toolbar, 'Tag', 'TOOL.View3D'), 'State', isequal(pf.get(
      'VIEW'), BrainSurfacePF.VIEW_3D_AZEL))
20         set(findobj(toolbar, 'Tag', 'TOOL.ViewSL'), 'State', isequal(pf.get(
      'VIEW'), BrainSurfacePF.VIEW_SL_AZEL))
21         set(findobj(toolbar, 'Tag', 'TOOL.ViewSR'), 'State', isequal(pf.get(
      'VIEW'), BrainSurfacePF.VIEW_SR_AZEL))
22         set(findobj(toolbar, 'Tag', 'TOOL.ViewAD'), 'State', isequal(pf.get(
      'VIEW'), BrainSurfacePF.VIEW_AD_AZEL))
23         set(findobj(toolbar, 'Tag', 'TOOL.ViewAV'), 'State', isequal(pf.get(
      'VIEW'), BrainSurfacePF.VIEW_AV_AZEL))
24         set(findobj(toolbar, 'Tag', 'TOOL.ViewCA'), 'State', isequal(pf.get(
      'VIEW'), BrainSurfacePF.VIEW_CA_AZEL))
25         set(findobj(toolbar, 'Tag', 'TOOL.ViewCP'), 'State', isequal(pf.get(
      'VIEW'), BrainSurfacePF.VIEW_CP_AZEL))
26     end
27 end
28 %%% igui!
29 pr = PanelPropRVectorView('EL', pf, 'PROP', BrainSurfacePF.VIEW, varargin
    {:});
30
31 %%% iprop!
32 ST_AXIS (figure, item) determines the axis settings.
33 %%% isettings!
34 'SettingsAxis'
35 %%% idefault!
36 SettingsAxis('GRID', false, 'AXIS', false)
37 %%% ipostset! ②
38 if pf.get('DRAWN')
39     toolbar = pf.get('H_TOOLBAR');
40     if check_graphics(toolbar, 'uitoolbar')
41         set(findobj(toolbar, 'Tag', 'TOOL.Grid'), 'State', pf.get('ST_AXIS')
      .get('GRID'))
42         set(findobj(toolbar, 'Tag', 'TOOL.Axis'), 'State', pf.get('ST_AXIS')
      .get('AXIS'))
43     end
44 end
45 %%% igui!
46 pr = SettingsAxisPP('EL', pf, 'PROP', BrainSurfacePF.ST_AXIS, varargin{:});
47
48 %%% iprop! ③
49 LISTENER_ST_AXIS (evanescent, handle) contains the listener to the axis
  settings to update the pushbuttons.
50 %%% icalculate!

```

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.

```

51 value = listener(pf.get('ST_AXIS'), 'PropSet', @cb_listener_st_axis);
52 %%% icalculate_callbacks!
53 function cb_listener_st_axis(~, ~)
54     if pf.get('DRAWN')
55         toolbar = pf.get('H_TOOLBAR');
56         if check_graphics(toolbar, 'uitoolbar')
57             set(findobj(toolbar, 'Tag', 'TOOL.Grid'), 'State', pf.get('
ST_AXIS').get('GRID'))
58             set(findobj(toolbar, 'Tag', 'TOOL.Axis'), 'State', pf.get('
ST_AXIS').get('AXIS'))
59         end
60     end
61 end
62 ...
63 %%% iprop!
64 BRAIN (figure, logical) determines whether the brain surface is shown.
65 %%% idefault!
66 true
67 %%% ipostset!
68 if pf.get('DRAWN')
69     if pf.get('BRAIN')
70         set(pf.get('H_BRAIN'), 'Visible', 'on')
71     else % ~pf.get('BRAIN')
72         set(pf.get('H_BRAIN'), 'Visible', 'off')
73     end
74
75     toolbar = pf.get('H_TOOLBAR'); ③
76     if check_graphics(toolbar, 'uitoolbar')
77         set(findobj(toolbar, 'Tag', 'TOOL.Brain'), 'State', pf.get('BRAIN'))
78     end
79 end
80 ...

```

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 toolbar pushbuttons for the brain surface. ← Code 5

```

1 %%% itests!
2
3 %%% iexcluded_props!
4 [BrainSurfacePF.PARENT BrainSurfacePF.H BrainSurfacePF.ST_POSITION
   BrainSurfacePF.ST_AXIS BrainSurfacePF.ST_SURFACE BrainSurfacePF.
   ST_ AMBIENT BrainSurfacePF.LISTENER_ST_AXIS] ①
5 ...

```

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.

```

1 %% iheader!
2 BrainAtlasPF < BrainSurfacePF (pf, panel figure brain atlas) is a plot of a
   brain atlas.
3
4 %%% idescription!
5 BrainAtlasPF manages the plot of the brain regions symbols,
6 spheres, ids and labels. BrainAtlasPF utilizes the surface created
7 from PFBrainSurface to integrate the regions to a brain surface.
8
9 %%% iseealso!
10 BrainAtlas, BrainSurface

```

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

```

1 %% iprops!
2
3 %%% iprop! ①
4 BA (metadata, item) is the brain atlas with the brain regions.
5 %%% isettings!
6 'BrainAtlas'
7
8 %%% iprop! ②
9 H.SPHS (evanescent, handlelist) is the set of handles for the spheres.
10 %%% icalculate! ③
11 L = pf.memorize('BA').get('BR_DICT').get('LENGTH');
12 h_sphs = cell(1, L);
13 for i = 1:L
14     h_sphs{i} = surf([], [], [], ...
15         'Parent', pf.memorize('H_AXES'), ...
16         'Tag', ['H_SPHS{' int2str(i) '}' ], ...
17         'Visible', false ...
18     );
19 end
20 value = h_sphs;
21
22 %%% iprop! ④
23 SPHS (figure, logical) determines whether the spheres are shown.
24 %%% idefault!
25 true
26 %%% ipostset!
27 if ~pf.get('SPHS') % false ⑤
28     h_sphs = pf.get('H_SPHS');
29     for i = 1:length(h_sphs)
30         set(h_sphs{i}, 'Visible', false)
31     end
32 else % true ⑥
33     % triggers the update of SPH_DICT

```

① contains the brain atlas to be visualized.

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

④ determines whether the spheres 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.

```

34     pf.set('SPH_DICT', pf.get('SPH_DICT'))
35 end
36
37 %%% iprop!
38 SPH_DICT (figure, idict) contains the spheres of the brain regions. ⑦
39 %%% isettings!
40 'SettingsSphere'
41 %%% ipostset!
42 if pf.get('SPHS') && ~isa(pf.get('BA'), 'NoValue') ⑧
43
44     br_dict = pf.get('BA').get('BR_DICT');
45
46     if pf.get('SPH_DICT').get('LENGTH') == 0 && br_dict.get('LENGTH') ⑨
47         for i = 1:1:br_dict.get('LENGTH')
48             br = br_dict.get('IT', i);
49             sphs{i} = SettingsSphere( ...
50                 'PANEL', pf, ...
51                 'PROP', BrainAtlasPF.H_SPHS, ...
52                 'I', i, ...
53                 'VISIBLE', true, ...
54                 'ID', br.get('ID'), ...
55                 'X', br.get('X'), ...
56                 'Y', br.get('Y'), ...
57                 'Z', br.get('Z'), ...
58                 'FACECOLOR', BRAPH2.COL, ...
59                 'FACEALPHA', 1 ...
60             );
61         end
62         pf.get('SPH_DICT').set('IT_LIST', sphs)
63     end
64
65     for i = 1:1:br_dict.get('LENGTH') ⑩
66         pf.get('SPH_DICT').get('IT', i).get('SETUP')
67     end
68
69     % reset the ambient lighting
70     pf.get('ST_AMBIENT').get('SETUP')
71 end
72 %%% igui! ⑪
73 pr = PanelPropIDictTable('EL', pf, 'PROP', BrainAtlasPF.SPH_DICT, ...
74     'COLS', [PanelPropIDictTable.SELECTOR SettingsSphere.VISIBLE
75             SettingsSphere.X SettingsSphere.Y SettingsSphere.Z SettingsSphere.
76             SPHERESIZE SettingsSphere.FACECOLOR SettingsSphere.FACEALPHA
77             SettingsSphere.EDGECOLOR SettingsSphere.EDGEALPHA], ...
78     varargin{:});
79
80
81 %%% iprops_update!
82 DRAW (query, logical) draws the figure brain atlas.
83 %%% icalculate!
84 value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:});
85 % also warning
86 if value
87     pf.memorize('H_SPHS') ⑫
88     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.

```

88
89 % reset the ambient lighting
90 pf.set('ST_AMBIENT').get('SETUP')
91 end
92
93 %%% iprop!
94 DELETE (query, logical) resets the handles when the panel figure brain
    surface is deleted.
95 %%% icalculate!
96 value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DELETE, varargin
    {:}); % also warning
97 if value
98     pf.set('H_SPHS', Element.getNoValue()) 14
99 end
100
101 %%% itests!
102
103 %%% iexcluded_props!
104 [BrainAtlasPF.PARENT BrainAtlasPF.H BrainAtlasPF.ST_POSITION BrainAtlasPF.
    ST_AXIS BrainAtlasPF.ST_SURFACE BrainAtlasPF.ST_AMBIENT]
105
106 ...

```

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](#)

```

1 %%% iprops!
2
3 ...
4
5 %%% iprop!
6 H_SYMS (evanescent, handlelist) is the set of handles for the symbols.
7 %%% icalculate!
8 L = pf.memorize('BA').get('BR_DICT').get('LENGTH');
9 h_syms = cell(1, L);
10 for i = 1:L
11     h_syms{i} = plot3(0, 0, 0, ...
12         'Parent', pf.get('H_AXES'), ...
13         'Tag', ['H_SYMS{' int2str(i) '}' ], ...
14         'Visible', false ...
15     );
16 end
17 value = h_syms;
18
19 %%% iprop!
20 SYMS (figure, logical) determines whether the symbols are shown.
21 %%% idefault!
22 false
23 %%% ipostset!
24 if ~pf.get('SYMS') % false
25     h_syms = pf.get('H_SYMS');
26     for i = 1:length(h_syms)
27         set(h_syms{i}, 'Visible', false)
28     end
29 else % true
30     % triggers the update of SYM_DICT
31     pf.set('SYM_DICT', pf.get('SYM_DICT'))
32 end
33
34 %%% iprop!

```

```

35 SYM_DICT (figure, idict) contains the symbols of the brain regions.
36 %%%% isettings!
37 'SettingsSymbol'
38 %%%% ipostset!
39 if pf.get('SYMS') && ~isa(pf.get('BA'), 'NoValue')
40
41     br_dict = pf.get('BA').get('BR_DICT');
42
43     if pf.get('SYM_DICT').get('LENGTH') == 0 && br_dict.get('LENGTH')
44         for i = 1:1:br_dict.get('LENGTH')
45             br = br_dict.get('IT', i);
46             syms{i} = SettingsSymbol( ...
47                 'PANEL', pf, ...
48                 'PROP', BrainAtlasPF.H_SYMS, ...
49                 'I', i, ...
50                 'VISIBLE', true, ...
51                 'ID', br.get('ID'), ... % Callback('EL', br, 'TAG', 'ID'),
52                 ...
53                 'X', br.get('X'), ... % Callback('EL', br, 'TAG', 'X'), ...
54                 'Y', br.get('Y'), ... % Callback('EL', br, 'TAG', 'Y'), ...
55                 'Z', br.get('Z') ... % Callback('EL', br, 'TAG', 'Z') ...
56             );
57         end
58         pf.get('SYM_DICT').set('IT_LIST', syms)
59     end
60     for i = 1:1:br_dict.get('LENGTH')
61         pf.get('SYM_DICT').get('IT', i).get('SETUP')
62     end
63 end
64 %%%% igui!
65 pr = PanelPropIDictTable('EL', pf, 'PROP', BrainAtlasPF.SYM_DICT, ...
66     'COLS', [PanelPropIDictTable.SELECTOR SettingsSymbol.VISIBLE
67             SettingsSymbol.X SettingsSymbol.Y SettingsSymbol.Z SettingsSymbol.
68             SYMBOL SettingsSymbol.SYMBOLSIZE SettingsSymbol.EDGECOLOR
69             SettingsSymbol.FACECOLOR], ...
70     varargin{:});
71
72 %%%% iprops_update!
73 ...
74
75 %%%% iprop!
76 DRAW (query, logical) draws the figure brain atlas.
77 %%%% icalculate!
78 value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:});
79 % also warning
80 if value
81     pf.memorize('H_SPHS')
82     pf.set('SPHS', pf.get('SPHS'))
83
84     pf.memorize('H_SYMS')
85     pf.set('SYMS', pf.get('SYMS'))
86
87     % reset the ambient lighting
88     pf.get('ST_AMBIENT').get('SETUP')
89 end
90
91 %%%% iprop!
92 DELETE (query, logical) resets the handles when the panel figure brain
93     surface is deleted.

```

```

90 %%% icalculate!
91 value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DELETE, varargin
    {:}); % also warning
92 if value
93     pf.set('H_SPHS', Element.getNoValue())
94     pf.set('H_SYMS', Element.getNoValue())
95 end
96
97 ...

```

Code 12: **BrainAtlasPF ids**. This code demonstrates how to add the ids to the BrainAtlasPF. ← [Code 11](#)

```

1 %% iprops!
2
3 ...
4
5 %%% iprop!
6 H_IDS (evanescent, handlelist) is the set of handles for the ids.
7 %%% icalculate!
8 L = pf.memorize('BA').get('BR_DICT').get('LENGTH');
9 h_ids = cell(1, L);
10 for i = 1:L
11     h_ids{i} = text(0, 0, 0, '', ...
12         'Parent', pf.get('H_AXES'), ...
13         'Tag', ['H_IDS{' int2str(i) '}'], ...
14         'Visible', false ...
15     );
16 end
17 value = h_ids;
18
19 %%% iprop!
20 IDS (figure, logical) determines whether the ids are shown.
21 %%% idefault!
22 false
23 %%% ipostset!
24 if ~pf.get('IDS') % false
25     h_ids = pf.get('H_IDS');
26     for i = 1:length(h_ids)
27         set(h_ids{i}, 'Visible', false)
28     end
29 else % true
30     % triggers the update of ID_DICT
31     pf.set('ID_DICT', pf.get('ID_DICT'))
32 end
33
34 %%% iprop!
35 ID_DICT (figure, idict) contains the ids of the brain regions.
36 %%% isettings!
37 'SettingsText'
38 %%% ipostset!
39 if pf.get('IDS') && ~isa(pf.get('BA'), 'NoValue')
40
41     br_dict = pf.get('BA').get('BR_DICT');
42
43     if pf.get('ID_DICT').get('LENGTH') == 0 && br_dict.get('LENGTH')
44         for i = 1:br_dict.get('LENGTH')
45             br = br_dict.get('IT', i);
46             ids{i} = SettingsText( ...
47                 'PANEL', pf, ...
48                 'PROP', BrainAtlasPF.H_IDS, ...

```



```

49         'I', i, ...
50         'VISIBLE', true, ...
51         'ID', br.get('ID'), ... % Callback('EL', br, 'TAG', 'ID'),
52     ...
53         'X', br.get('X'), ... % Callback('EL', br, 'TAG', 'X'), ...
54         'Y', br.get('Y'), ... % Callback('EL', br, 'TAG', 'Y'), ...
55         'Z', br.get('Z'), ... % Callback('EL', br, 'TAG', 'Z'), ...
56         'TXT', br.get('ID') ... % Callback('EL', br, 'TAG', 'ID')
57     ...
58 );
59 end
60 pf.get('ID_DICT').set('IT_LIST', ids)
61 end
62 for i = 1:1:br_dict.get('LENGTH')
63     pf.get('ID_DICT').get('IT', i).get('SETUP')
64 end
65 end
66 %%% igui!
67 pr = PanelPropIDictTable('EL', pf, 'PROP', BrainAtlasPF.ID_DICT, ...
68     'COLS', [PanelPropIDictTable.SELECTOR SettingsText.VISIBLE SettingsText.
69         X SettingsText.Y SettingsText.Z SettingsText.ROTATION SettingsText.TXT
70         SettingsText.FONTNAME SettingsText.FONTSIZE SettingsText.FONTCOLOR
71         SettingsText.INTERPRETER], ...
72     varargin{:});
73 %%% iprops_update!
74 ...
75 %%% iprop!
76 DRAW (query, logical) draws the figure brain atlas.
77 %%% icalculate!
78 value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:});
79 % also warning
80 if value
81     pf.memorize('H_SPHS')
82     pf.set('SPHS', pf.get('SPHS'))
83
84     pf.memorize('H_SYMS')
85     pf.set('SYMS', pf.get('SYMS'))
86
87     pf.memorize('H_IDS')
88     pf.set('SPHS', pf.get('SPHS'))
89
90     % reset the ambient lighting
91     pf.get('ST_AMBIENT').get('SETUP')
92 end
93 %%% iprop!
94 DELETE (query, logical) resets the handles when the panel figure brain
95 surface is deleted.
96 %%% icalculate!
97 value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DELETE, varargin
98     {:}); % also warning
99 if value
100     pf.set('H_SPHS', Element.getNoValue())
101     pf.set('H_SYMS', Element.getNoValue())
102     pf.set('H_IDS', Element.getNoValue())
103 end
104
105

```

Code 13: **BrainAtlasPF labels**. This code demonstrates how to add the labels to the BrainAtlasPF. ← [Code 12](#)

```

102 ...
1 %% iprops!
2
3 ...
4
5 %%% iprop!
6 H_LABS (evanescent, handlelist) is the set of handles for the labels.
7 %%% icalculate!
8 L = pf.memorize('BA').get('BR_DICT').get('LENGTH');
9 h_labs = cell(1, L);
10 for i = 1:L
11     h_labs{i} = text(0, 0, 0, '', ...
12         'Parent', pf.get('H_AXES'), ...
13         'Tag', ['H_LABS{' int2str(i) '}''], ...
14         'Visible', false ...
15     );
16 end
17 value = h_labs;
18
19 %%% iprop!
20 LABS (figure, logical) determines whether the labels are shown.
21 %%% idefault!
22 false
23 %%% ipostset!
24 if ~pf.get('LABS') % false
25     h_labs = pf.get('H_LABS');
26     for i = 1:length(h_labs)
27         set(h_labs{i}, 'Visible', false)
28     end
29 else % true
30     % triggers the update of LAB_DICT
31     pf.set('LAB_DICT', pf.get('LAB_DICT'))
32 end
33
34 %%% iprop!
35 LAB_DICT (figure, idict) contains the labels of the brain regions.
36 %%% isettings!
37 'SettingsText'
38 %%% ipostset!
39 if pf.get('LABS') && ~isa(pf.get('BA'), 'NoValue')
40
41     br_dict = pf.get('BA').get('BR_DICT');
42
43     if pf.get('LAB_DICT').get('LENGTH') == 0 && br_dict.get('LENGTH')
44         for i = 1:br_dict.get('LENGTH')
45             br = br_dict.get('IT', i);
46             labs{i} = SettingsText( ...
47                 'PANEL', pf, ...
48                 'PROP', BrainAtlasPF.H_LABS, ...
49                 'I', i, ...
50                 'VISIBLE', true, ...
51                 'ID', br.get('ID'), ... % Callback('EL', br, 'TAG', 'ID'),
52
53                 ...
54                 'X', br.get('X'), ... % Callback('EL', br, 'TAG', 'X'), ...
55                 'Y', br.get('Y'), ... % Callback('EL', br, 'TAG', 'Y'), ...
56                 'Z', br.get('Z'), ... % Callback('EL', br, 'TAG', 'Z'), ...

```

```

55         'TXT', br.get('LABEL') ... % Callback('EL', br, 'TAG', '
        LABEL') ...
56     );
57     end
58     pf.get('LAB_DICT').set('IT_LIST', labs)
59 end
60
61 for i = 1:1:br_dict.get('LENGTH')
62     pf.get('LAB_DICT').get('IT', i).get('SETUP')
63 end
64 end
65 %%% igui!
66 pr = PanelPropIDictTable('EL', pf, 'PROP', BrainAtlasPF.LAB_DICT, ...
67     'COLS', [PanelPropIDictTable.SELECTOR SettingsText.VISIBLE SettingsText.
        X SettingsText.Y SettingsText.Z SettingsText.ROTATION SettingsText.TXT
        SettingsText.FONTNAME SettingsText.FONTSIZE SettingsText.FONTCOLOR
        SettingsText.INTERPRETER], ...
68     varargin{:});
69
70 %% iprops_update!
71
72 ...
73
74 %%% iprop!
75 DRAW (query, logical) draws the figure brain atlas.
76 %%% icalculate!
77 value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:});
    % also warning
78 if value
79     pf.memorize('H_SPHS')
80     pf.set('SPHS', pf.get('SPHS'))
81
82     pf.memorize('H_SYMS')
83     pf.set('SYMS', pf.get('SYMS'))
84
85     pf.memorize('H_IDS')
86     pf.set('SPHS', pf.get('SPHS'))
87
88     pf.memorize('H_LABS')
89     pf.set('LABS', pf.get('LABS'))
90
91     % reset the ambient lighting
92     pf.get('ST_AMBIENT').get('SETUP')
93 end
94
95 %%% iprop!
96 DELETE (query, logical) resets the handles when the panel figure brain
    surface is deleted.
97 %%% icalculate!
98 value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DELETE, varargin
    {:}); % also warning
99 if value
100     pf.set('H_SPHS', Element.getNoValue())
101     pf.set('H_SYMS', Element.getNoValue())
102     pf.set('H_IDS', Element.getNoValue())
103     pf.set('H_LABS', Element.getNoValue())
104 end
105
106 ...

```

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](#)

```

1 %% iprops!
2
3 ...
4
5 %% iprop!
6 SPHS (figure, logical) determines whether the spheres are shown.
7 %%%% idefault!
8 true
9 %%%% ipostset!
10 if ~pf.get('SPHS') % false
11     h_sphs = pf.get('H_SPHS');
12     for i = 1:length(h_sphs)
13         set(h_sphs{i}, 'Visible', false)
14     end
15 else % true
16     % triggers the update of SPH_DICT
17     pf.set('SPH_DICT', pf.get('SPH_DICT'))
18 end
19
20 % update state of toggle tool
21 toolbar = pf.get('H_TOOLBAR');
22 if check_graphics(toolbar, 'uitoolbar')
23     set(findobj(toolbar, 'Tag', 'TOOL.Sphs'), 'State', pf.get('SPHS'))
24 end
25
26 ...
27
28 %% iprop!
29 SYMS (figure, logical) determines whether the symbols are shown.
30 %%%% idefault!
31 false
32 %%%% ipostset!
33 if ~pf.get('SYMS') % false
34     h_syms = pf.get('H_SYMS');
35     for i = 1:length(h_syms)
36         set(h_syms{i}, 'Visible', false)
37     end
38 else % true
39     % triggers the update of SYM_DICT
40     pf.set('SYM_DICT', pf.get('SYM_DICT'))
41 end
42
43 % update state of toggle tool
44 toolbar = pf.get('H_TOOLBAR');
45 if check_graphics(toolbar, 'uitoolbar')
46     set(findobj(toolbar, 'Tag', 'TOOL.Syms'), 'State', pf.get('SYMS'))
47 end
48
49 ...
50
51 %% iprop!
52 IDS (figure, logical) determines whether the ids are shown.
53 %%%% idefault!

```

```

54 false
55 %%%% ipostset!
56 if ~pf.get('IDS') % false
57     h_ids = pf.get('H_IDS');
58     for i = 1:length(h_ids)
59         set(h_ids{i}, 'Visible', false)
60     end
61 else % true
62     % triggers the update of ID_DICT
63     pf.set('ID_DICT', pf.get('ID_DICT'))
64 end
65
66 % update state of toggle tool
67 toolbar = pf.get('H_TOOLBAR');
68 if check_graphics(toolbar, 'uitoolbar')
69     set(findobj(toolbar, 'Tag', 'TOOL.Ids'), 'State', pf.get('IDS'))
70 end
71
72 ...
73
74 %%%% iprop!
75 LABS (figure, logical) determines whether the labels are shown.
76 %%%% idefault!
77 false
78 %%%% ipostset!
79 if ~pf.get('LABS') % false
80     h_labs = pf.get('H_LABS');
81     for i = 1:length(h_labs)
82         set(h_labs{i}, 'Visible', false)
83     end
84 else % true
85     % triggers the update of LAB_DICT
86     pf.set('LAB_DICT', pf.get('LAB_DICT'))
87 end
88
89 % update state of toggle tool
90 toolbar = pf.get('H_TOOLBAR');
91 if check_graphics(toolbar, 'uitoolbar')
92     set(findobj(toolbar, 'Tag', 'TOOL.Labs'), 'State', pf.get('LABS'))
93 end
94
95 ...
96
97 %%% iprops_update!
98
99 ...
100
101 %%%% iprop!
102 DRAW (query, logical) draws the figure brain atlas.
103 %%%% icalculate!
104 value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.DRAW, varargin{:});
105     % also warning
106 if value
107     pf.memorize('H_SPHS')
108     pf.set('SPHS', pf.get('SPHS')) % sets also    SPH_DICT
109
110     pf.memorize('H_SYMS')
111     pf.set('SYMS', pf.get('SYMS')) % sets also    SYM_DICT
112
113     pf.memorize('H_IDS')
114     pf.set('SPHS', pf.get('SPHS')) % sets also    ID_DICT

```

```

114
115     pf.memorize('H_LABS')
116     pf.set('LABS', pf.get('LABS')) % sets also LAB_DICT
117
118     % reset the ambient lighting
119     pf.get('ST_AMBIENT').get('SETUP')
120 end
121 %%% icalculate_callbacks!
122 function cb_sphs(~, ~, sphs) % (src, event)
123     pf.set('SPHS', sphs)
124 end
125 function cb_syms(~, ~, syms) % (src, event)
126     pf.set('SYMS', syms)
127 end
128 function cb_ids(~, ~, ids) % (src, event)
129     pf.set('IDS', ids)
130 end
131 function cb_labs(~, ~, labs) % (src, event)
132     pf.set('LABS', labs)
133 end
134
135 ...
136
137 %%% iprop!
138 H_TOOLS (evanescent, handlelist) is the list of panel-specific tools from
    the first.
139 %%% icalculate!
140 toolbar = pf.memorize(PanelFig.H_TOOLBAR);
141 if check_graphics(toolbar, 'uitoolbar')
142     value = calculateValue@BrainSurfacePF(pf, BrainSurfacePF.H_TOOLS);
143
144     tool_separator_1 = uipushtool(toolbar, 'Separator', 'on', 'Visible', '
        off');
145
146     % Spheres
147     tool_sphs = uitoggletool(toolbar, ...
148         'Tag', 'T00L.Sphs', ...
149         'Separator', 'on', ...
150         'State', pf.get('SPHS'), ...
151         'Tooltip', 'Show Spheres', ...
152         'CData', imread('icon_sphere.png'), ...
153         'OnCallback', {@cb_sphs, true}, ...
154         'OffCallback', {@cb_sphs, false});
155
156     % Symbols
157     tool_syms = uitoggletool(toolbar, ...
158         'Tag', 'T00L.Syms', ...
159         'Separator', 'on', ...
160         'State', pf.get('SYMS'), ...
161         'Tooltip', 'Show Symbols', ...
162         'CData', imread('icon_symbol.png'), ...
163         'OnCallback', {@cb_syms, true}, ...
164         'OffCallback', {@cb_syms, false});
165
166     % IDs
167     tool_ids = uitoggletool(toolbar, ...
168         'Tag', 'T00L.Ids', ...
169         'Separator', 'on', ...
170         'State', pf.get('IDS'), ...
171         'Tooltip', 'Show IDs', ...
172         'CData', imread('icon_id.png'), ...

```

```

173         'OnCallback', {cb_ids, true}, ...
174         'OffCallback', {cb_ids, false});
175
176     % Labels
177     tool_labs = uitoggletool(toolbar, ...
178         'Tag', 'TOOL.Labs', ...
179         'Separator', 'on', ...
180         'State', pf.get('LABS'), ...
181         'Tooltip', 'Show Labels', ...
182         'CData', imread('icon_label.png'), ...
183         'OnCallback', {cb_labs, true}, ...
184         'OffCallback', {cb_labs, false});
185
186     value = {value{:}}, ...
187         tool_separator_1, ...
188         tool_sphs, tool_syms, tool_ids, tool_labs ...
189     };
190 else
191     value = {};
192 end
193
194 . . .

```
