General Developer Tutorial for BRAPH 2.0

The BRAPH 2 Developers

October 15, 2023

The software architecture of BRAPH 2.0 provides a clear structure for developers to understand and extend its functionalities. All objects in BRAPH 2.0 are derived from a base object called Element. Developers can easily add new elements such as brain surfaces, atlases, example scripts, GUI pipelines, graphs, measures, data types, data importers, data exporters, and analyses. By writing new elements and recompiling the code, the new elements and their functionalities are immediately integrated also into the GUI. In this Developer Tutorial, we will explain how BRAPH 2.0 is compiled, how the elements are strcutured, and finally we will provide examples of how new elements can be implemented.

Contents

Genesis

explain genesis process and concept of elements with pseudo code

Need for two compilations and hardcoding

braph2genesis

regenerate + explain when recompile necessary

Elements

BRAPH 2.0 is a compiled object-oriented programming software. The base class for all elements is `Element`. Each element is essentially a container for a series of *properties*. Each propery has a *category* and a *format*. The category determines for what and how a property. The possible categories are shown in the box below. Even though it is possible to create instances of Element, typically one uses its subclasses. In this section, we will see how to implement a new element.

Overview

Element, NoValue, Callback, Concrete Element + FIG

Property Categories

`CONSTANT` Static constant equal for all instances of the element. It allows incoming callbacks.

`METADATA` Metadata NOT used in the calculation of the results. It does not allow callbacks. It is not locked when a result is calculated.

`PARAMETER` Parameter used to calculate the results of the element. It allows incoming and outgoing callbacks. It is connected with a callback when using a template. It is locked when a result is calculated.

`DATA` Data used to calculate the results of the element. It is NoValue when not set. It allows incoming and outgoing callbacks. It is locked when a result is calculated.

`RESULT` Result calculated by the element using parameters and data. The calculation of a result locks the element. It is NoValue when not calculated. It allows incoming callbacks.

`QUERY` Query result calculated by the element. The calculation of a query does NOT lock the element. It is NoValue when not calculated. It does not allow callbacks.

`EVANESCENT` Evanescent variable calculated at runtime (typically employed for handles of GUI components). It is NoValue when not calculated. It does not allow callbacks.

`FIGURE` Parameter used to plot the results in a figure. It allows incoming and outgoing callbacks. It is not locked when a result is calculated.

`GUI` Parameter used by the graphical user interface (GUI). It allows incoming and outgoing callbacks. It is not locked when a result is calculated.

Property Categories

`EMPTY` Empty has an empty value and is typically used as a result or query to execute some code.

`STRING` String is a char array.

`STRINGLIST` StringList is a cell array with char arrays.

`LOGICAL` Logical is a boolean value.

`OPTION` Option is a char array representing an option within a set defined in the element (case sensitive).
Settings: cell array of chars representing the options, e.g., `{'plus', 'minus', 'zero'}`.

`CLASS` Class is a char array corresponding to an element class.
Settings: class name of a subclass of Element (or Element itself).

`CLASSLIST` ClassList is a cell array with char arrays corresponding to element classes.
Settings: class name of a subclass of Element (or Element itself), which represents the base element.

`ITEM` Item is a pointer to an element of a class defined in the element.
Settings: class name of a subclass of Element (or Element itself).

`ITEMLIST` ItemList is a cell array with pointers to elements of a class defined in the element.
Settings: class name of a subclass of Element (or Element itself), which represents the base element.

`IDICT` Idict is an indexed dictionary of elements of a class defined in the element.
Settings: class name of a subclass of Element (or Element itself), which represents the dictionary element.

`SCALAR` Scalar is a scalar numerical value.

`RVECTOR` RVector is a numerical row vector.

`CVECTOR` CVector is a numerical column vector.

`MATRIX` Matrix is a numerical matrix.

`SMATRIX` SMatrix is a numerical square matrix.

`CELL` Cell is a 2D cell array of numeric data, typically used for adjacency matrices and measures.

`NET` Net is a MatLab neural network object (network, SeriesNetwork, DAGNetwork, dlnetwork).

`HANDLE` Handle is a handle for a graphical or listener component. It should only be used as an evanescent property.

`HANDLELIST` HandleList is a cell array with handles for graphical or listener components. It should only be used as an evanescent property.

`COLOR` Color is an RGB color, e.g., `'[1 0 0]'` for red.

`ALPHA` Alpha is a transparency level between 0 and 1.

`SIZE` Size represents the size of a graphical componet. It is a positive number (default = 1).

`MARKER` Marker represents the marker style. It can be `'o'`, `'+'`, `'*'`, `'.'`,'x', `'_'`, `'|'`, `'s'`, `'d'`, `''̂`, `'v'`, `'>'`, `'<'`, `'p'`, `'h'`, '' (no marker).

`LINE` Line represents the line style. It can be `'-'`, `':'`, `'-.'`, `'-'`, '' (no line).

Property lifecycle: get, set, memorize, lock

Box all tokens

Box special tokens

Overview of Elements

The core code includes the compiler (`genesis`), the essential source code (`src`), and the GUI functionalities (`gui`).

Explain element structure + FIG

Implementation of an Element

Concrete Element extension

basic header and basic props

show how it works, set, get, memorize, lock

Results, Data, Parameters

locking, seeded randomness

Query

Evanescent, Gui, Figure