

Create a New BRAPH 2 Distribution: Hello, World!

The BRAPH 2 Developers

February 13, 2025

The software architecture of BRAPH 2 provides a clear structure for developers to understand and extend its functionalities. Developers can easily create new distributions with their own pipelines. By recompiling BRAPH 2, the new pipelines and their functionalities are integrated in a new distribution with its own graphical user interface. In this developer tutorial, you will learn how to create a new distribution of BRAPH 2 that contains a simple pipeline where the user enters a first string (e.g., "Hello") and a second string (e.g., "World"), and the pipeline combines them into a single string ("Hello World!").

Contents

<i>Create New Distribution Folder</i>	2
<i>Create New Elements</i>	2
<i>_StringUnit.gen.m</i>	3
<i>_StringCombine.gen.m</i>	5
<i>Create New Pipeline</i>	8
<i>Create Configuration File</i>	9
<i>Compile New Distribution</i>	10
<i>Run Example "Hello, World!"</i>	10

Create New Distribution Folder

First, we need to create a new folder that will contain all the new scripts referring to this new distribution. For the example explained in this tutorial the name of this new folder will be `helloworld`. Furthermore, the genesis compiler from the Standard BRAPH 2 Distribution needs to be downloaded. This compiler is a Matlab file ([braph2genesis.m](#)) located in the Standard BRAPH 2 Github Repository. This file needs to be placed outside the new created folder `helloworld` as shown in Figure 1.

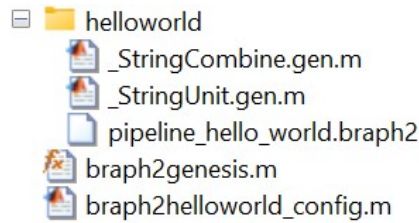


Figure 1: **Folder tree.** All files needed to create the new "Hello, World!" BRAPH 2 Distribution.

Create New Elements

BRAPH 2 is a compiled object-oriented programming software. Its objects are elements, which contain a set of properties of various categories and formats, as described in detail in the tutorial [General Developer Tutorial for BRAPH 2](#)). All objects are derived from a base object called `Element` and written in a simplified pseudocode `*.gen.m` that is compiled into the actual elements (files `*.m`) by the command `braph2genesis`. Even though it is possible to create instances of `Element`, it does not have any props and typically one uses its subclasses. Its three direct subclasses are `NoValue`, `Callback`, and `ConcreteElement`. The details of each subclass can be found in tutorial [General Developer Tutorial for BRAPH 2](#)).

In this tutorial it is used the subclass `ConcreteElement` that provides the infrastructure necessary for all concrete elements, like strings. Thus, following these rules the following two new objects need to be created and placed inside the previously created folder `helloworld` as shown in Figure 1.:

- **`_StringUnit.gen.m`**: contains a user-defined string.
- **`_StringCombine.gen.m`**: combines the strings from two string units.

The new generator files will have the same structure: `iheader!` , `ilayout!`, `jprops` (properties), `jprops_update!` (update the properties of

the ConcreteElement) and itests! (to add unit tests). While header! the token ;build! are required, the rest is optional.

_StringUnit.gen.m

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

```

1  %% iheader!
2  StringUnit < ConcreteElement (s, string) contains a user-defined string.
3
4  ①
5
6  %% idescription!
7  A String Unit (StringUnit) contains a user-defined string.
8  This element is created for distribution demonstration purpose.
9
10 %% iseealso!
11 StringCombine, PanelPropString
12
13 %% ibuild!
14 1

```

① defines StringUnit as a subclass of ConcreteElement. The moniker will be S.

Code 2: **StringUnit layout.** The layout section of the generator code in *_StringUnit.gen.m* provides the information about the layout that we want to create for the StringUnit element.

```

1  %% ilayout! ①
2
3  %% iprop!
4  %%% iid!
5  StringUnit.S
6  %%% ititle!
7  Specified String
8
9
10

```

① creates a new layout in the graphical user interface allowing the user to introduce a string

Code 3: **StringUnit element props update.** The props_update section of the generator code in *_StringUnit.gen.m* updates the properties of the ConcreteElement element.

```

1  %% iprops_update!
2
3  %% iprop!
4  ELCLASS (constant, string) is the class of the string unit.
5  %%% idefault!
6  'StringUnit'
7
8  %% iprop!
9  NAME (constant, string) is the name of the string unit.
10 %%% idefault!
11 'String Unit'
12
13 %% iprop!

```

```

14 DESCRIPTION (constant, string) is the description of the string unit.
15 %%%% idefault!
16 'A String Unit (StringUnit) contains a user-defined string. This element
    is created for distribution demonstration purpose.'
17
18 %%% iprop!
19 TEMPLATE (parameter, item) is the template of the string unit.
20 %%%% isettings!
21 'StringUnit'
22
23 %%% iprop!
24 ID (data, string) is a few-letter code for the string unit.
25 %%%% idefault!
26 'StringUnit ID'
27
28 %%% iprop!
29 LABEL (metadata, string) is an extended label of the string unit.
30 %%%% idefault!
31 'StringUnit label'
32
33 %%% iprop!
34 NOTES (metadata, string) are some specific notes about the string unit.
35 %%%% idefault!
36 'StringUnit notes'
37

```

Code 4: **StringUnit element props.** The props section of generator code in `_StringUnit.gen.m` defines the properties to be used in `StringUnit`.

```

1  %% iprops!
2
3  %%% iprop!
4  S (data, string) is the user-defined string.
5  %%%% idefault!
6  'Hello'
7

```

Code 5: **StringUnit element tests.** The tests section of generator code in `_StringUnit.gen.m`

```

1  %% itests!
2
3  %%% itest!
4  %%%% iname!
5  test
6  %%%% icode!
7  defined_string = 'test';
8  su = StringUnit('S', defined_string);
9
10 assert(isequal(su.get('S'), defined_string), ...
11 [BRAPH2.STR ':StringUnit:' BRAPH2.FAIL_TEST], ...
12 'StringUnit does not store the defined string properly.' ...
13 )
14

```

_StringCombine.gen.m

The `_StringCombine.gen.m` generator file is created the same way as `_StringUnit.gen.m`.

Code 6: StringCombine element header. The header section of the generator code in `_StringCombine.gen.m` provides the general information about the `StringCombine` element.

```

1  %% iheader!
2  StringCombine < ConcreteElement (sc, string combine) combines the string
3    from two string units.
4
5  %% idescription!
6  A String Combine (StringCombine) combines the strings from two string
7    units.
8  This element is created for distribution demonstration purpose.
9  ①
10
11 %% iseealso!
12 StringUnit, PanelPropString
13
14 %% ibuild!
15 1

```

① defines `StringCombine` as a subclass of `ConcreteElement`. The moniker will be `sc`.

Code 7: StringCombine layout. The layout section of the generator code in `_StringCombine.gen.m` provides the information about the layout that we want to crete for the `StringCombine` element.

```

1  %% ilayout!
2
3  %% iprop!
4  %%% iid!
5  StringCombine.SU1
6  %%% ititle!
7  The First String
8
9  %% iprop!
10 %%% iid!
11 StringCombine.SU2
12 %%% ititle!
13 The Second String
14
15 %% iprop!
16 %%% iid!
17 StringCombine.S_COMBINED
18 %%% ititle!
19 Combined Strings
20
21

```

Code 8: StringCombine element props update. The `props_update` section of the generator code in `_StringCombinet.gen.m` updates the properties of the `ConcreteElement` element.

```

1  %% iprops_update!
2
3  %% iprop!
4  ELCLASS (constant, string) is the class of the string combine.
5  %%% idefault!
6  'StringCombine'
7
8  %% iprop!
9  NAME (constant, string) is the name of the string combine.
10 %%% idefault!
11 'String Combine'
12
13 %% iprop!
14 DESCRIPTION (constant, string) is the description of the string combine.
15 %%% idefault!
16 'A String Combine (StringCombine) combines the strings from two string
    units. This element is created for distribution demonstration purpose.'
17
18 %% iprop!
19 TEMPLATE (parameter, item) is the template of the string combine.
20 %%% isettings!
21 'StringCombine'
22
23 %% iprop!
24 ID (data, string) is a few-letter code for the string combine.
25 %%% idefault!
26 'StringCombine ID'
27
28 %% iprop!
29 LABEL (metadata, string) is an extended label of the string combine.
30 %%% idefault!
31 'StringCombine label'
32
33 %% iprop!
34 NOTES (metadata, string) are some specific notes about the string combine.
35 %%% idefault!
36 'StringCombine notes'
37

```

Code 9: **StringCombine element props.** The props section of generator code in `_StringCombine.gen.m` defines the properties to be used in `StringCombine`.

```

1  %% iprops!
2
3  %% iprop!
4  SU1 (data, item) is the first string unit.
5  %%% isettings!
6  'StringUnit'
7
8  %% iprop!
9  SU2 (data, item) is the second string unit.
10 %%% isettings!
11 'StringUnit'
12
13 %% iprop!
14 S_COMBINED (result, string) is the combined strings.
15 %%% icalculate!
16 value = [sc.get('SU1').get('S') ' ' sc.get('SU2').get('S')]; ①
17

```

① combines the strings which are specified with SU1 and SU2 from a `StringUnit` element.

18

Code 10: **StringCombine** element tests. The tests section of generator code in `_StringCombine.gen.m`

```

1  %% itests!
2
3  %% itest!
4  %%% iname!
5  test
6  %%% icode!
7  defined_string1 = 'test1';
8  su1 = StringUnit('S', defined_string1);
9
10 defined_string2 = 'test2';
11 su2 = StringUnit('S', defined_string2);
12
13 sc = StringCombine('SU1', su1, 'SU2', su2);
14
15 assert(isequal(sc.get('S_COMBINED'), [defined_string1 ' ' defined_string2
16   ]), ...
17   [BRAPH2.STR ':StringCombine:' BRAPH2.FAIL_TEST], ...
18   'StringCombine does not combine the defined strings properly.' ...
19 ) ①

```

① we verify that the new element `StringCombine` successfully combines the strings which are specified with `su1` and `su2`

Create New Pipeline

Once the new elements are created, a pipeline that takes the created elements and runs the desired functionality is needed. In this case the pipeline will combine two strings (e.g. "Hello") and (e.g. "World") and combine them (e.g. "Hello, World!").

Code 11: Pipeline "Hello, World!" We need to create a new file (e.g., pipeline_hello_world.braph2) that includes the following code to create a new pipeline for the "Hello, World!" example.

```

1  %% Pipeline Hello World
2
3  % This pipeline script demonstrates a 'Hello World' example of the BRAPH 2
4    Hello World distribution.
5  % 1. It defines the first string, with a default value of 'Hello'.
6  % 2. It defines the second string, with a default value of 'World'.
7  % 3. It combines both strings, resulting in the default output 'Hello
8    World'.
9
10 % PDF:
11 % README:
12
13 %% String 1 ①
14 su1 = StringUnit('S', 'Hello'); % Define the First String % First String
15
16 %% String 2 ①
17 su2 = StringUnit('S', 'World'); % Define the Second String % Second String
18
19 %% Combine Strings
20 sc = StringCombine('SU1', su1, 'SU2', su2); % Combine Strings % First
21    String + Second String

```

① By default 'Hello' and 'World' can be set. The user will be able to modify the string as desired

Create Configuration File

A configuration file is needed to build a new distribution.

Code 12: **Configuration file** We need to create a new file (e.g., `braph2helloworld_config.m`) with all the configuration details.

```

1  distribution_name = 'Hello, World!'; ①
2  distribution_moniker = 'helloworld'; ②
3  pipeline_folders = {
4      'helloworld', ...
5  }; ③
6  braph2_version = 'heads/ycw-lite-genesis'; ④
7
8  % Add here all included and excluded folders and elements ⑤
9  % '-folder'          the folder and its elements will be excluded
10 %
11 % '+folder'         the folder is included, but not its elements
12 % '+_ElementName.gen.m' the element is included,
13 %                   if the folder is included
14 %
15 % '+folder*'        the folder and its elements are included
16 % '+_ElementName.gen.m' the element is excluded,
17 %                   if the folder and its elements are included
18 % (by default, the folders are included as '+folder*')
19 rollcall = { ...
20     'util', '+_Exporter.gen.m', '+_Importer.gen.m', ...
21     '+ds*', '-ds_examples', ...
22     '-analysis', ...
23     '-atlas', ...
24     '-gt', ...
25     '-cohort', ...
26     '-nn', ...
27     '+gui', '+_ExporterPipelineBRAPH2.gen.m', '+_GUI.gen.m', '+_GUIElement.
28     gen.m', ...
29     '+_GUIFig.gen.m', '+_GUILayout.gen.m', '+_ImporterPipelineBRAPH2.gen.m',
30     ...
31     '+_Panel.gen.m', '+_PanelElement.gen.m', '+_PanelFig.gen.m', '+
32     _PanelProp.gen.m', ...
33     '+_PanelPropItem.gen.m', '+_PanelPropString.gen.m', '+_Pipeline.gen.m',
34     '+_PipelineCode.gen.m', ...
35     '+_PipelinePP_Notes.gen.m', '+_PipelinePP_PSDict.gen.m', '+
36     _PipelineSection.gen.m', ...
37     '+_PanelPropStringTextArea.gen.m', ...
38     '-brainsurfs', ...
39     '-atlases', ...
40     '-graphs', ...
41     '-measures', ...
42     '-neuralnetworks', ...
43     '+pipelines', ...
44     '+helloworld*', '+hellouniverse*', ...
45     '+test*', ...
46     '-sandbox' ...
47 }; ⑥
48
49 files_to_delete = { ...
50     ['src' filesep 'gui' filesep 'test_GUIFig.m'], ...
51     ['src' filesep 'gui' filesep 'test_PanelFig.m'], ...
52 };

```

① specifies the name of the new distribution. This name will appear in the main window of the new distribution GUI

② short identifier used in launcher functions and filenames. It should only contain letters, numbers, and underscores.

③ defines the folders that contain the newly created pipelines we want to include in our new distribution.

④ defines the version of BRAPH 2 (e.g., 'tags/2.0.1' or 'heads/develop') to fetch from Github to create the new distribution. The build number should be 7 or larger (version 2.0.1 or subsequent).

⑤ defines which folders and elements from the Standard BRAPH 2 distribution we want to maintain in our new distribution. With this definition we avoid compiling elements that are not necessary and that involve long compilation times.

⑥ specifies files to remove after compilation

Compile New Distribution

To compile the software, you simply just need to run Code 14.

Code 13: **Execute BRAPH 2 compiler.** To compile the newly created (or new version) distribution software you need to call the BRAPH 2 genesis compiler specifying the configuration file with the details of the new distribution.

```
1 braph2genesis('braph2helloworld_config.m')
```

If compilation ended successfully the following log message should appear *****All good!*****. A new braph2helloworld folder would have been generated which can then be used as explained in Section . **WARNING:** To ensure a successful compilation, the folder braph2helloworld (if already created) should be erased and all of its dependencies should be removed from the MATLAB path. This folder will be regenerated after a successful compilation.

Run Example "Hello, World!"

To run the software, you simply just need to run Code 14 and a new GUI will appear as shown in Figure 2.

Code 14: **Execute BRAPH 2 compiler.** To run the newly created (or new version) distribution software you need to call the newly created launcher file.

```
1 braph2helloworld
```

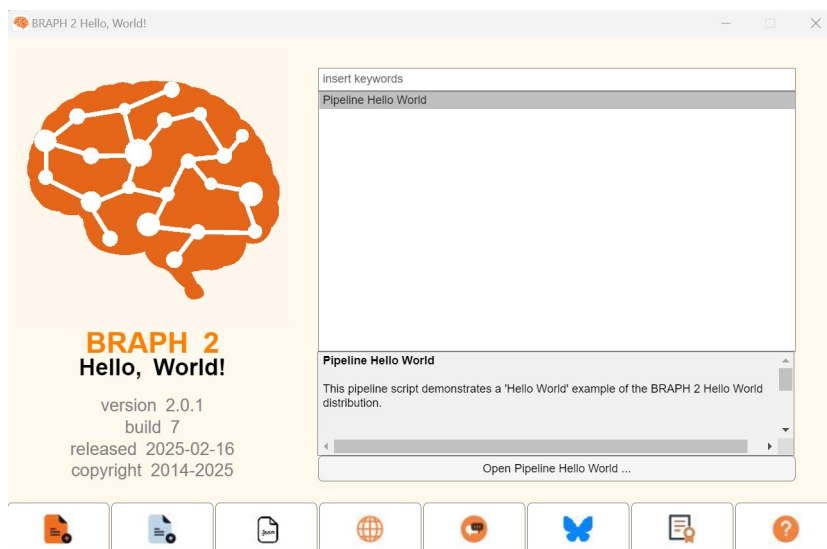


Figure 2: New BRAPH 2 "Hello, World!" distribution.. This is the newly created GUI for the BRAPH 2 "Hello, World!" distribution

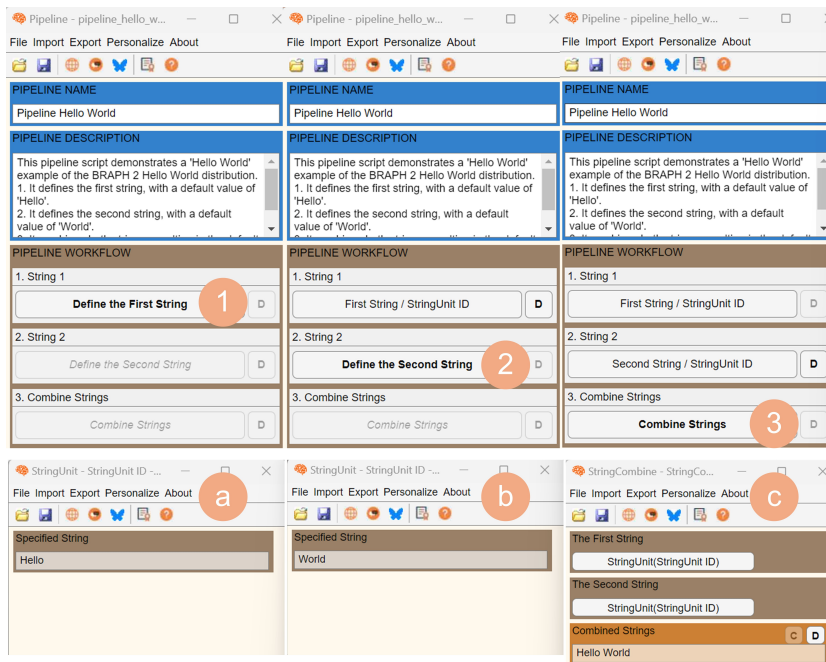


Figure 3: Running "Hello, World!" Example.. Steps to run the example case: **a** Click on Define the First String from the pipeline GUI. A new GUI window will open **a**. Here you can introduce whatever string ("Hello" by default). **2** Click on Second String from the pipeline GUI. A new GUI window will open **b**. Here you can introduce whatever string ("World" by default). **c** Click on Combine Strings to obtain the combination of the two previously defined strings A new GUI window will open **b**. Here you need to press **c** to obtained the combined strings.