Create a New BRAPH 2 Distribution: Hello, World!

The BRAPH 2 Developers

February 12, 2025

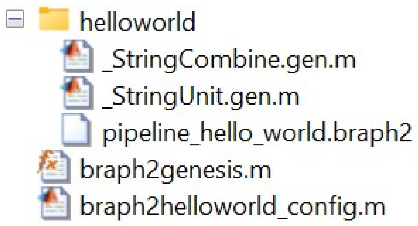
The software architecture of BRAPH 2 provides a clear structure for developers to understand and extend its functionalities. Developers can easily create new distributions with their own pipelines. By recompiling BRAPH 2, the new pipelines and their functionalities are integrated in a new distribution 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 q first string (e.g., "Hello") and a second string (e.g., "World"), and the pipeline combines them into a single string ("Hello World!").

Contents

Create New Distribution Folder 2
Create New Elements 2
  _StringUnit.gen.m 3
  _StringCombine.gen.m 5
Create New Pipeline 8
Create Configuration File 9
Compile New Distribution 11
Run Example "Hello, World!" 11

Create New Distribution Folder

First, we need to create a new folder where we will have all the new scripts referring to this new distribution. For the example explained in this tutorial we will name this new folder `helloworld`. Furthermore, we need to download the genesis compiler from the Standard BRAPH 2 Distribution. This compiler is a matlab file braph2genesis.m) located in the Standard BRAPH 2 Github Repository. We will place this file outside the new created folder `helloworld` as shown in Figure 1.



```
helloworld
    _StringCombine.gen.m
    _StringUnit.gen.m
    pipeline_hello_world.braph2
braph2genesis.m
braph2helloworld_config.m
```

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: `¡header!` , `¡layout!`, *¡props* (properties), *¡props_update!* (update the properties of the ConcreteElement) and ¡tests! (to add unit tests). header! is required (and the token ¡build!), while 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.

```
%% ¡header!
StringUnit < ConcreteElement (s, string) contains a user-defined string.

(1)

%%% ¡description!
A String Unit (StringUnit) contains a user-defined string.
This element is created for distribution demonstration purpose.

%%% ¡seealso!
StringCombine, PanelPropString

%%% ¡build!
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 crete for the StringUnit element.

```

%% ¡layout!   (1)

%%% ¡prop!
%%%% ¡id!
StringUnit.S
%%%% ¡title!
Specified String


```

(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.

```
%% ¡props_update!

%%% ¡prop!
ELCLASS (constant, string) is the class of the string unit.
%%%% ¡default!
'StringUnit'

%%% ¡prop!
NAME (constant, string) is the name of the string unit.
%%%% ¡default!
'String Unit'

%%% ¡prop!
```

```
DESCRIPTION (constant, string) is the description of the string unit.
%%%% ¡default!
'A String Unit (StringUnit) contains a user-defined string. This element
   is created for distribution demonstration purpose.'

%%% ¡prop!
TEMPLATE (parameter, item) is the template of the string unit.
%%%% ¡settings!
'StringUnit'

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

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

%%% ¡prop!
NOTES (metadata, string) are some specific notes about the string unit.
%%%% ¡default!
'StringUnit notes'

```

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

```
%% ¡props!

%%% ¡prop!
S (data, string) is the user-defined string.
%%%% ¡default!
'Hello'

```

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

```
%% ¡tests!

%%% ¡test!
%%%% ¡name!
test
%%%% ¡code!
defined_string = 'test';
su = StringUnit('S', defined_string);

assert(isequal(su.get('S'), defined_string), ...
[BRAPH2.STR ':StringUnit:' BRAPH2.FAIL_TEST], ...
'StringUnit does not store the defined string properly.' ...
)

```

_StringCombine.gen.m

We will create the `StringCombine` generator file the same way as we did with `StringUnit`.

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

```

%% iheader!
StringCombine < ConcreteElement (sc, string combine) combines the string
   from two string units.

%%% idescription!
A String Combine (StringCombine) combines the strings from two string
   units.
This element is created for distribution demonstration purpose.

(1)

%%% iseealso!
StringUnit, PanelPropString

%%% ibuild!
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.

```

%% ilayout!

%%% iprop!
%%%% iid!
StringCombine.SU1
%%%% ititle!
The First String

%%% iprop!
%%%% iid!
StringCombine.SU2
%%%% ititle!
The Second String

%%% iprop!
%%%% iid!
StringCombine.S_COMBINED
%%%% ititle!
Combined Strings

```

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.

```
%% iprops_update!

%%% iprop!
ELCLASS (constant, string) is the class of the string combine.
%%%% idefault!
'StringCombine'

%%% iprop!
NAME (constant, string) is the name of the string combine.
%%%% idefault!
'String Combine'

%%% iprop!
DESCRIPTION (constant, string) is the description of the string combine.
%%%% idefault!
'A String Combine (StringCombine) combines the strings from two string
   units. This element is created for distribution demonstration purpose.'

%%% iprop!
TEMPLATE (parameter, item) is the template of the string combine.
%%%% isettings!
'StringCombine'

%%% iprop!
ID (data, string) is a few-letter code for the string combine.
%%%% idefault!
'StringCombine ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the string combine.
%%%% idefault!
'StringCombine label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the string combine.
%%%% idefault!
'StringCombine notes'

```

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

```
%% iprops!

%%% iprop!
SU1 (data, item) is the first string unit.
%%%% isettings!
'StringUnit'

%%% iprop!
SU2 (data, item) is the second string unit.
%%%% isettings!
'StringUnit'

%%% iprop!
S_COMBINED (result, string) is the combined strings.
%%%% icalculate!
value = [sc.get('SU1').get('S') ' ' sc.get('SU2').get('S')]; (1)

```

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

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

```
%% ¡tests!

%%% ¡test!
%%%% ¡name!
test
%%%% ¡code!
defined_string1 = 'test1';
su1 = StringUnit('S', defined_string1);

defined_string2 = 'test2';
su2 = StringUnit('S', defined_string2);

sc = StringCombine('SU1', su1, 'SU2', su2);

assert(isequal(sc.get('S_COMBINED'), [defined_string1 ' ' defined_string2
   ]), ...
[BRAPH2.STR ':StringCombine:' BRAPH2.FAIL_TEST], ...
'StringCombine does not combine the defined strings properly.' ...
) ①

```

① we verify that the new element `StringCombine` succesfully combines the strings which are specified with `sU1` and `sU2`

Create New Pipeline

Once we have created our new elements we create a pipeline that takes the created elements and runs the desired functionality, which in this case is combining the two strings "Hello" and "World" and combine them into "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.

```
%% Pipeline Hello World

% This pipeline script demonstrates a 'Hello World' example of the BRAPH 2
    Hello World distribution.
% 1. It defines the first string, with a default value of 'Hello'.
% 2. It defines the second string, with a default value of 'World'.
% 3. It combines both strings, resulting in the default output 'Hello
   World'.

% PDF:
% README:

%% String 1
su1 = StringUnit('S', 'Hello'); % Define the First String % First String

%% String 2
su2 = StringUnit('S', 'World'); % Define the Second String % Second String

%% Combine Strings
sc = StringCombine('SU1', su1, 'SU2', su2); % Combine Strings % First
   String + Second String
```

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.

```
distribution_name = 'Hello, World!'; ①
distribution_moniker = 'helloworld'; ②
pipeline_folders = {
  'helloworld', ...
};③
braph2_version = 'heads/ywc-lite-genesis'; ④

% Add here all included and excluded folders and elements ⑤
% '-folder'                 the folder and its elements will be excluded
%
% '+folder'                 the folder is included, but not its elements
%   '+_ElementName.gen.m'   the element is included,
%                           if the folder is included
%
% '+folder*'                the folder and its elements are included
%   '-_ElementName.gen.m'   the element is excluded,
%                           if the folder and its elements are included
% (by default, the folders are included as '+folder*')
rollcall = { ...
  '+util', '+_Exporter.gen.m', '+_Importer.gen.m', ...
  '+ds*', '-ds_examples', ...
  '-analysis', ...
  '-atlas', ...
  '-gt', ...
  '-cohort', ...
  '-nn', ...
  '+gui', '+_ExporterPipelineBRAPH2.gen.m', '+_GUI.gen.m', '+_GUIElement.gen.m', ...
  '+_GUIFig.gen.m', '+_GUILayout.gen.m', '+_ImporterPipelineBRAPH2.gen.m', ...
  '+_Panel.gen.m', '+_PanelElement.gen.m', '+_PanelFig.gen.m', '+_PanelProp.gen.m', ...
  '+_PanelPropItem.gen.m', '+_PanelPropString.gen.m', '+_Pipeline.gen.m', '+_PipelineCode.gen.m', ...
  '+_PipelinePP_Notes.gen.m', '+_PipelinePP_PSDict.gen.m', '+_PipelineSection.gen.m', ...
  '+_PanelPropStringTextArea.gen.m', ...
  '-brainsurfs', ...
  '-atlases', ...
  '-graphs', ...
  '-measures', ...
  '-neuralnetworks', ...
  '+pipelines', ...
  '+helloworld*', '+hellouniverse*', ...
  '+test*', ...
  '-sandbox' ...
};

files_to_delete = { ...
  ['src' filesep 'gui' filesep 'test_GUIFig.m'], ...
  ['src' filesep 'gui' filesep 'test_PanelFig.m'], ...
};⑥
```

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 an 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 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.

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

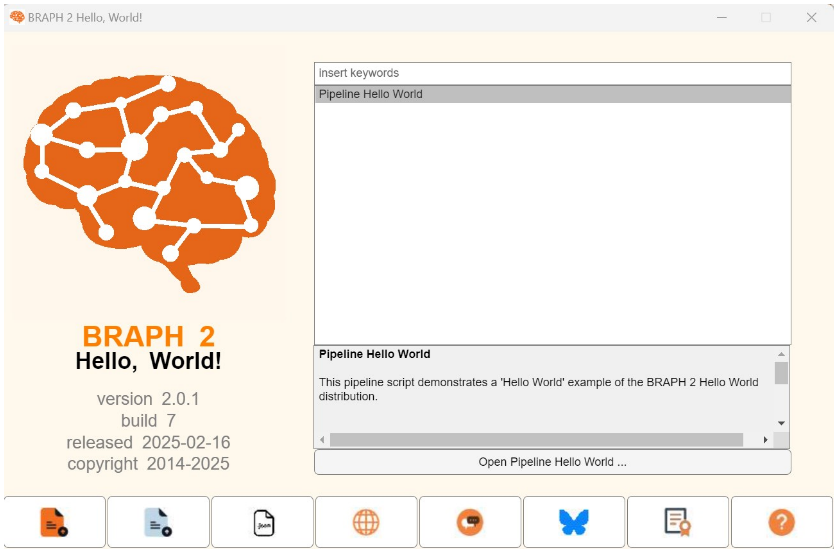
If compilation ended successfully the following log message should appear `***All good!***`. A new `braph2helloworld` folder would have been generated which can then be used as explained in Section . WARNING: To ensure a successful compilation, the folder `braph2helloworld` (if already created) should 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.

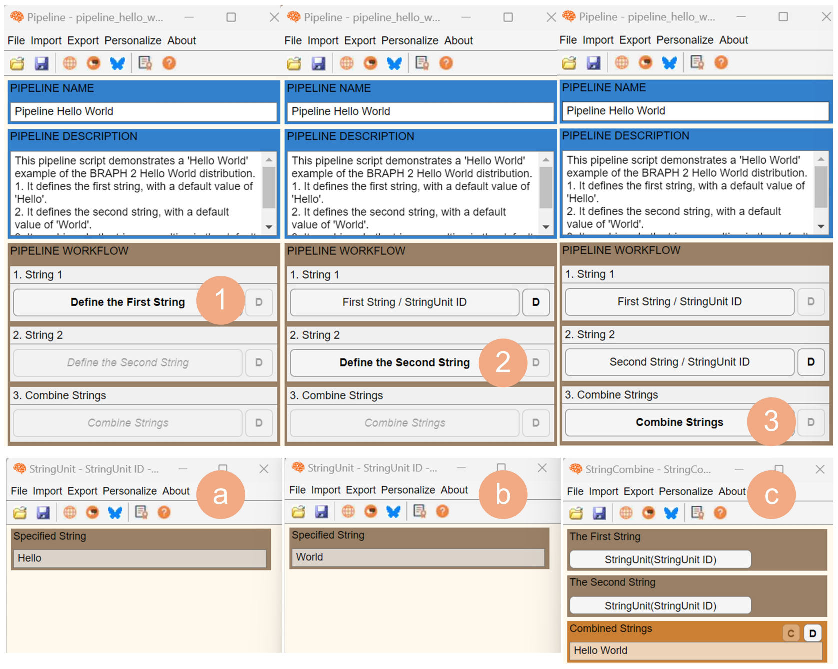
Code 14: **Execute BRAPH 2 compiler.** To run the newly created (or new version) distribution software you need to call the newly created launcher file.

```
braph2helloworld
```



This pipeline script demonstrates a 'Hello World' example of the BRAPH 2 Hello World distribution.

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


PIPELINE NAME

Pipeline Hello World

PIPELINE DESCRIPTION

This pipeline script demonstrates a 'Hello World' example of the BRAPH 2 Hello World distribution.
1. It defines the first string, with a default value of 'Hello'.
2. It defines the second string, with a default value of 'World'.

Specified String: Hello

Specified String: World

The First String: StringUnit(StringUnit ID)

The Second String: StringUnit(StringUnit ID)

Combined Strings: Hello World

Pipeline - pipeline_hello_w...
File Import Export Personalize About
PIPELINE WORKFLOW
1. String 1
Define the First String
1
D
2. String 2
Define the Second String
D
3. Combine Strings
Combine Strings
D
First String / StringUnit ID
2
Second String / StringUnit ID
3
StringUnit - StringUnit ID -...
a
b
StringCombine - StringCo...
c
C
D

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.