Documentation for the BML Experiment Toolkit

Author: Adam Bebko

Contents

[About 1](#_Toc865827)

[Overview 1](#_Toc865828)

# About

The BML Experiment Toolkit helps you design and run experiments in unity quickly and iteratively without fussing over coding details. You define your variables and experiment structure, and the toolkit will automatically create a table of trials to run. You can customize what happens before, during, and after a trial, block of trials, or the experiment itself, while not having to worry about the details of setting up your experiment. The toolkit will go through all the trials, outputting the results automatically to a file.

The toolkit adds a menu called BML Toolkit, which contains several useful items for running your experiments.

# Overview

The main structure of an experiment is defined with a variable configuration file. In this file you define your independent and dependent variables, and any blocking or randomization that needs to occur.

Figure 1: Example of a basic variable configuration file

Once the variable structure has been defined, you need to write a script to define what occurs during a trial. You create a script inheriting from the toolkit’s base class and overwrite functions that are called automatically during the experiment. For each “stage” of the experiment (Experiment, Block, Trial), you can define custom behavior that occurs pre-stage, during the stage, and post-stage. Examples of such customization might include a “welcome screen” before the experiment, a “thank you” screen after the experiment, a pause between each block of trials, and instructions before each trial. These customizations can be written very quickly.

The main requirement for setting up an experiment is defining what occurs during a trial. A trial script has access to each of the variables you defined in the config file, so you can set up objects in your scene using values from the independent variable, or write results to the dependent variables, which will be output for each trial into a text file.

Figure 2: Basic structure of an automatically generated experiment. You can customize functionality for all functions except grey ones. You must define the Main() function for your trials to define what should happen in each trial.

After hitting play in unity, a new experimental session is created, and a window appears prompting you for session details such as participant ID, and which block order to use.

The trial structure is displayed to double-check correctness, and then you can begin the experiment. The experiment begins running, calling the code you defined above.

# Configuring your experimental design

With the toolkit, you need only define your variables, and your experimental design will be created automatically for you.

To set up your experimental design you need to define your variables. Create a new Config file from the BML Toolkit menu. It will appear in your project’s Assets folder.

To add a variable, select your desired data type (int, string, etc.), type of variable (independent, dependent), and then click on Create Variable. You should see your variable added to the list below. Configure your variable by naming it, adding values, selecting whether the variable will be used to create blocks of trials, etc.

To delete a variable from the list click the Delete Variable button

Your changes will be saved automatically.

When complete, drag your config file into your Custom Experiment GameObject Prefab, and it will run.

The power of the configuration files is that you can run the same experiment with multiple saved configurations. For example, you can select different values for your experiment in different config files and drag your desired configuration into the experiment object prior to running participants. This also allows you to iteratively design your experiments while saving previous configurations.

## Variable options

#### Name

You must name your variables. This will be how you access their values within your trials. I recommend using one-word names, or joining words using underscores. Do not use any special characters or punctuation.

#### Datatype

Once created, you cannot modify this variable’s datatype. This lets Unity know what kind of data to expect. To get a value from a variable in a trial you must know its datatype to avoid errors. Currently, supported datatypes include:

* Int
* Float
* String
* GameObject
* Vector3

#### Variable Type

Independent and dependent variables are treated differently. Independent variables define your experimental structure, and dependent variables are values that determined from responses or measurements during each trial

### Independent variable options:

#### Block

Enabling this option will create blocks of trials for each value defined. Useful for counterbalancing or variables that require setup in real life (i.e. participants need to move between values)

#### Mixing Type of Variable

This option will define how the variable is mixed with the other variables when creating your experimental design.

**Balanced:**

This will create trials for every combination of values of each balanced variable. Example: for two 3-value balanced variables, there will be 3x3=9 trials.

|  |  |  |
| --- | --- | --- |
| Trial Number | Balanced Variable 1 | Balanced Variable 2 |
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 1 | 3 |
| 4 | 2 | 1 |
| 5 | 2 | 2 |
| 6 | 2 | 3 |
| 7 | 3 | 1 |
| 8 | 3 | 2 |
| 9 | 3 | 3 |

**Looped:**

This will loop through the values such that there is an equal number of trials with each value. Lowest common multiple.

Eample1: For a variable with 2 values, and another variable with 4 values, the following table will be created. In this case the lowest common multiple is 4 trials.

|  |  |  |
| --- | --- | --- |
| Trial Number | Looped Variable 1 | Looped Variable 2 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 1 | 3 |
| 4 | 2 | 4 |

Example2: For a variable with 2 values, and another variable with 3 values the following table. In this case the lowest common multiple is 6 trials.

|  |  |  |
| --- | --- | --- |
| Trial Number | Looped Variable 1 | Looped Variable 2 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 1 | 3 |
| 4 | 2 | 1 |
| 5 | 1 | 2 |
| 6 | 2 | 3 |

**Even Probability:**

Each trial will have a randomly selected value for this variable, with the same probability for each value.

Example, A balanced variable with 6 levels, and an even probability variable with 10 values (numbers 1-10)

|  |  |  |
| --- | --- | --- |
| Trial Number | Balanced Variable 1 | Even Probability Variable 2 |
| 1 | 1 | 4 |
| 2 | 2 | 8 |
| 3 | 3 | 9 |
| 4 | 4 | 4 |
| 5 | 5 | 7 |
| 6 | 6 | 2 |

**Custom Probability:**

Each trial will have a randomly selected value for this variable, with a defined probability of being selected. You define the probability to the right of each value (as a decimal). The final probability is automatically calculated to ensure they add up to 1.

Example, A balanced variable with 6 levels, and a custom probability variable with 2 values with the first value having 0.2 probability, and the second value having 0.8 probability

|  |  |  |
| --- | --- | --- |
| Trial Number | Balanced Variable 1 | Custom Probability Variable 2 |
| 1 | 1 | 2 |
| 2 | 2 | 2 |
| 3 | 3 | 2 |
| 4 | 4 | 1 |
| 5 | 5 | 2 |
| 6 | 6 | 2 |

#### Values

Define the levels of each variable. You must define at least one value.

#### Probability

Only visible when custom probability mixing type is selected. This is where you define your probability of each value being picked.

### Dependent variable options

#### Default value

This is the value given to your dependent variables in case there is no response given, or the variable is not updated in your trial. This can be left blank if desired.

# Session

A session defines one time through an experiment. It stores options that affect the whole experiment. You can set options such as debug mode for testing out your experiment, the participant’s ID, block order number and the output file.

You can change your session settings when running your experiment program. By default, the session settings will display your previously used settings for reference.

Dug mode will save your output into a debug file created in your project’s Assets/Debug/debugFile. It can also be used to change functionality in your trials for testing purposes.

You can manually name your output files, or let the toolkit do it for you based on the current date, time, and participant ID.

This is where you choose your desired block order or have it randomly select the block order for you.

# Experiment

The Experiment GameObject Prefab is your main window into running experiments with the toolkit. To Define an experiment, you need to create a new script and have it inherit from the Experiment class.

public class MyFancyExperiment : Experiment {

}

Within this class you can define custom behavior to occur pre- and post- experiment. To do so, you write the following

public class MyFancyExperiment : Experiment {

protected override IEnumerator Post() {

//Your code here

return null;

}

protected override IEnumerator Pre() {

//Your code here

return null;

}

}

Lets try to understand this a bit.

## A Note on Coroutines and IEnumerators

Normal loops in code keep looping until done, and don’t allow for you to extend code across multiple frames in a unity program.

For example, the following code while hang your program since it never lets the program get to the next frame.

void NormalKindOfMethod() {

while (true) {

//do something

}

}

What Coroutines and IEnumerators let you do is to have it loop, but then pause each loop to let the rest of the program run. Then, the next frame of your program it will pick up where it left off. Therefore your program can continue to run normally, and this loop gets accessed each frame.

IEnumerator CoroutineMethod() {

while (true) {

//do something

yield return null; //continue below this after next frame

Debug.Log("this will print to console the next frame");

}

}

To begin a coroutine method, you cannot call it like a normal method. You need to use a special unity function called StartCoroutine. This is already written for you in the toolkit, so you can rest assured your Pre() and Post() methods are called before and after your experiment is run. You must only tell Unity if you want it to pause to wait for the next frame. You can call normal methods inside coroutine functions, and everything else behaves as you would expect it to, with the added power of being able to wait for the next frame.

Instead of waiting for one frame, you can have unity wait for a speicified amount of time before continuing. This is great for displaying instructions. Or creating time delays in your code. You can have unity display some instructions to a participant, wait for a few seconds, then stop displaying the instructions. Below is an example of displaying instructions at the start of your program for 5 seconds.

protected override IEnumerator Pre() {

DisplayStartingInstructions(); //called right away

//the rest of your program will run normally while this waits.

yield return new WaitForSeconds(5);

StopDisplayingInstructions(); //will only get called after 5 seconds

}

void DisplayStartingInstructions() {

//your code for displaying

}

void StopDisplayingInstructions() {

//your code for stopping to display

}