

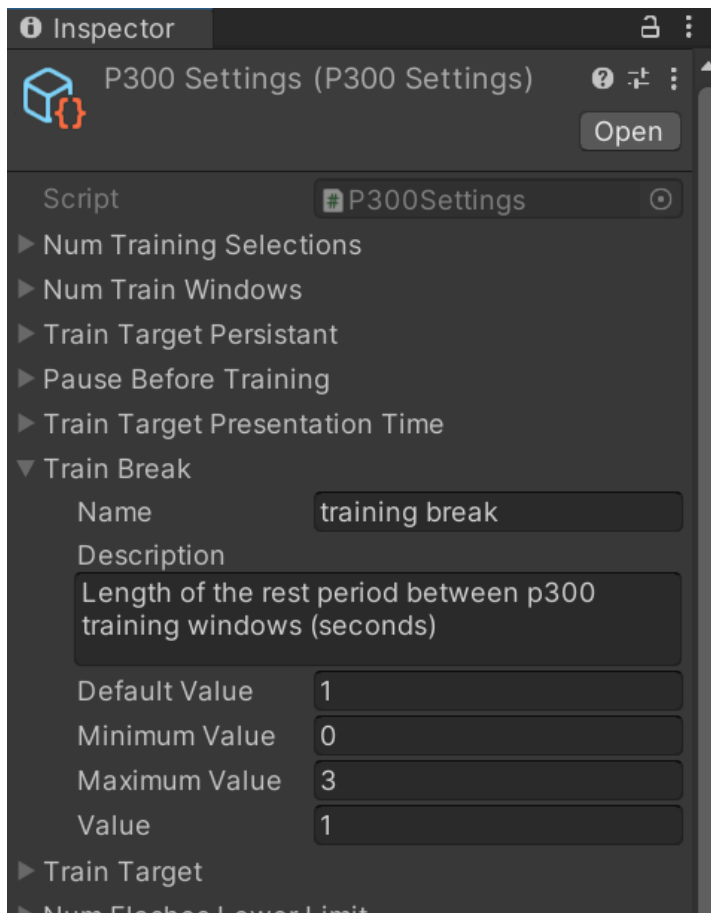
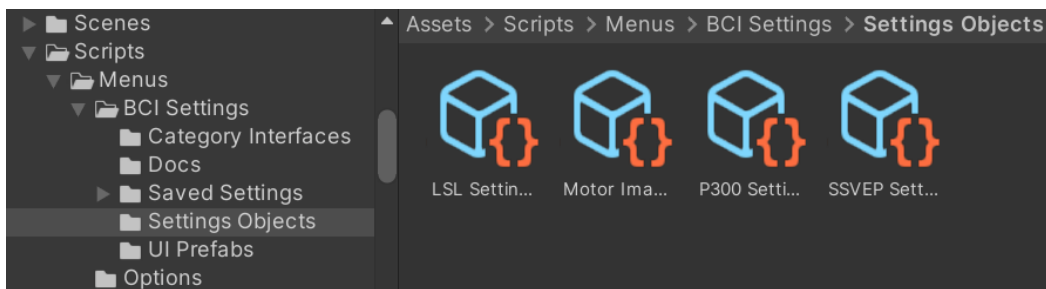
How to Add, Remove, and Edit BCI Settings

The system handling BCI settings exists to provide a human readable settings file and remain functional as the list of settings it saves is altered. This document aims to provide much more detail than necessary, but please reach out with any questions.

Editing Extant Settings (click the objects, change the stuff)

Settings are organized into categories according to their intended “destination”. Settings that affect the P300 Controller Behavior are in P300 settings and so on. Each category has a scriptable object in:

Assets/Scripts/Menus/BCI Settings/Settings Objects.



From these scriptable objects you can alter the display name, default value and suggested range of each setting.

The default value, range, and description can be altered seamlessly and will be immediately reflected by the settings file itself the next time it is saved.

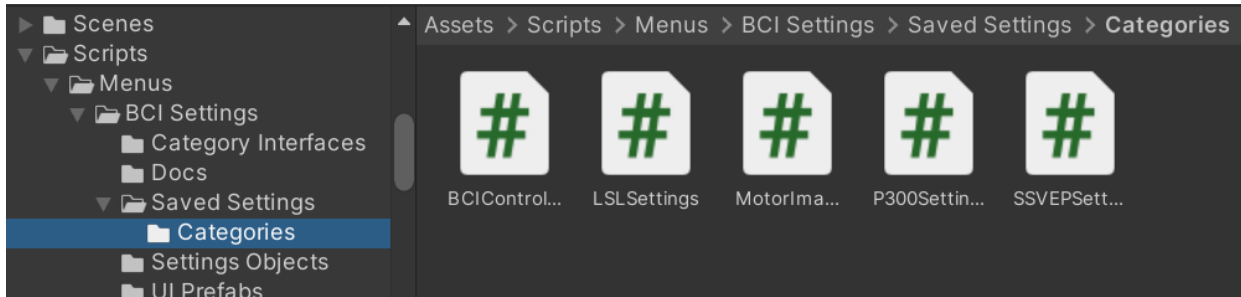
Changing the display name will cause some warning statements to be printed (lost connection to saved value) but will also work seamlessly in practice.

The current value of the setting is only displayed as an in engine utility and should really just be ignored.

Adding or Removing Settings

To add or remove settings, the definitions of the category must be edited in code. These definitions can all be found in:

Assets/Scripts/Menus/BCI Settings/Saved Settings/Categories



To add or remove a setting, there are 3 steps:

- Add or remove member variable
 - The constructor is completely unnecessary, just preference
- Add or remove the variable from the enumerator
- Add the application functionality

```
6 public class MotorImagerySettings : BCIControllerSettings<MIControllerBehavior>
7 {
8     3 references
9     public override string Name => "Motor Imagery Settings";
10
11     public IntegerSetting numSelectionsBeforeTraining = new("selections before training", 3, 2, 5);
12     public IntegerSetting numSelectionsBetweenTraining = new("selections between training", 3, 2, 5);
13
14     11 references
15     public override IEnumerator<SettingBase> GetEnumerator()
16     {
17         IEnumerator<SettingBase> baseIE = base.GetEnumerator();
18         while (baseIE.MoveNext())
19         {
20             yield return baseIE.Current;
21         }
22         yield return numSelectionsBeforeTraining;
23         yield return numSelectionsBetweenTraining;
24     }
25
26     4 references
27     public override void ApplyToController(MIControllerBehavior target)
28     {
29         base.ApplyToController(target);
30         target.numSelectionsBeforeTraining = numSelectionsBeforeTraining;
31         target.numSelectionsBetweenTraining = numSelectionsBetweenTraining;
32     }
33 }
```

Handwritten red annotations in the code block: A red '1' and a red curly brace '}' are next to the constructor lines (11-12). A red asterisk '*' is next to the base enumerator line (17). A red '2' and a red curly brace '}' are next to the yield return lines (22-23). A red '3' and a red curly brace '}' are next to the target assignment lines (30-31).

* These sections include settings from the base class

Each of the functional settings categories (all but LSL), inherit from a base BCIControllerSettings class that shares training attributes across all relevant categories. Edit this base class to add or remove settings from this shared pool.

```
8 public class BCIControllerSettings<ControllerType> : SettingsBlock where ControllerType: BCIControllerBehavior
9 {
10     5 references
11     public override string Name => "BCI Controller Settings";
12
13     public IntegerSetting numTrainingSelections = new("training selection count", 9, 6, 12);
14     public IntegerSetting numTrainWindows = new("training window count", 3, 2, 5);
15
16     public ToggleSetting trainTargetPersistant = new("train target persistant", false);
17     public FloatSetting pauseBeforeTraining = new("pause before training", 2, 1, 5);
18     public FloatSetting trainTargetPresentationTime = new("training target presentation time", 3, 2, 5);
19     public FloatSetting trainBreak = new("training break", 1, 0, 3);
20
21     public IntegerSetting trainTarget = new("training target", 99, 8, 99);
22
23     12 references
24     public override IEnumerator<SettingBase> GetEnumerator()
25     {
26         yield return numTrainingSelections;
27         yield return numTrainWindows;
28         yield return trainTargetPersistant;
29         yield return pauseBeforeTraining;
30         yield return trainTargetPresentationTime;
31         yield return trainBreak;
32         yield return trainTarget;
33     }
34
35     5 references
36     public virtual void ApplyToController(ControllerType target)
37     {
38         target.numTrainingSelections = numTrainingSelections;
39         target.numTrainWindows = numTrainWindows;
40         target.trainTargetPersistant = trainTargetPersistant;
41         target.pauseBeforeTraining = pauseBeforeTraining;
42         target.trainTargetPresentationTime = trainTargetPresentationTime;
43         target.trainBreak = trainBreak;
44         target.trainTarget = trainTarget;
45     }
46 }
```

Adding or Editing Categories

This should be pretty straightforward to figure out, but you might be better off reaching out to bridge the step of connecting it to the in game menu.