Руководство по программированию логических блоков Vizio

Логический блок (Logic Block) – это класс, унаследованный от класса LogicBlock, находящегося в пространстве имен Antares.Vizio.Runtime

Каждый логический блок может иметь:

1. Входные триггеры (Entry Trigger).
2. Выходные триггеры (Output Triggers).
3. Переменные, которые делятся на:
    a. Входные (In)
    b. Выходные (Out)
    c. Внутренние (Internal)

Входные триггеры – это функции, выполняющие действия. Действия чаще всего бывают следующими:

- Обработка входных переменных (например, вызов функции).
- Назначение значений выходным переменным.
- Активация выходных триггеров.

Выходные триггеры – это средство передачи управления другим логическим блокам.

Переменные – данные, с которыми работают логические блоки.

Чтобы создать логический блок достаточно где-то в проекте создать скрипт, в скрипте объявить класс и унаследовать его от Antares.Vizio.Runtime.LogicBlock. Описать и реализовать его составляющие (триггеры и переменные). Чтобы система нашла блок и поместила его в панель инструментов (Toolbox) необходимо созданный класс пометить атрибутом VisualLogicBlock, в котором указать название блока и его местоположение.

После перекомпиляции проекта Vizio автоматически найдет данный блок, и его можно будет использовать в своих визуальных программах.

Пример простого логического блока, выполняющего вычисление функции Sin:

```
using UnityEngine;
using Antares.Vizio.Runtime;

[VisualLogicBlock("My Sin Block", "Smart Blocks")]
public class MySinBlock : LogicBlock
{
  [Parameter(VariableType.In, typeof(float))]
  public Variable f;

  [Parameter(VariableType.Out, typeof(float))]
  public Variable result;

  public override void OnInitializeDefaultData()
  {
     RegisterOutputTrigger("Exit");
  }

  [EntryTrigger]
  public void In()
  {
    result.Value = Mathf.Sin((float)f.Value);

    ActivateTrigger("Exit");
  }
}
```

Итак, по порядку:

```
using UnityEngine;
using Antares.Vizio.Runtime;
```

Эти две строки импортируют два пространства имен, в которых находятся необходимые типы данных.

```
[VisualLogicBlock("My Sin Block", "Smart Blocks")]
public class MySinBlock : LogicBlock
```

Здесь выполняется объявление класса MyBlockSin как наследника LogicBlock. Класс помечен атрибутом, VisualLogicBlock, параметрами которого являются имя блока (“My Sin Block ”) и его местоположение (“Smart Blocks ”).

```
  [Parameter(VariableType.In, typeof(float))]
  public Variable f;
```

```
[Parameter(VariableType.Out, typeof(float))]
public Variable result;
```

Здесь выполняется объявление двух переменных блока. Каждая переменная должна иметь тип Variable и быть публичной (public). Так же, чтобы среда обработала переменную – ее необходимо пометить атрибутом Parameter. Этот атрибут указывает среде – как использовать данную переменную. Первый параметр атрибута определяет назначение переменной и может принимать следующие варианты значений:

- VariableType.In – входная переменная (сохраняет свое значение при редактировании визуального графа логики)
- VariableType.Out – выходная переменная (не сохраняет своего значения во время редактирования, должна быть заполнена значением при выполнении действия блоком)
- VariableType.Internal – внутренняя переменная (сохраняет значение, может быть использована для хранения каких-либо данных, может редактироваться программно, через пользовательский инспектор блока (CustomInspector))

Второй параметр атрибута указывает тип данных переменной. В нашем примере функция Sin получает угол в радианах, который является вещественным (float), результат функции так же вещественное число (float).

Класс Variable имеет свойство Value, в котором находится значение переменной, свойство имеет тип object (System.Object) и для получения значения необходимо вручную приводить тип переменной (это не касается js).

```
public override void OnInitializeDefaultData()
{
   RegisterOutputTrigger("Exit");
}
```

Каждый блок при создании получает событие OnInitializeDefaultData, в этом событии необходимо выполнить инициализацию начального состояния блока. В первую очередь это касается выходных триггеров. Чтобы зарегистрировать выходной триггер – необходимо вызвать функцию RegisterOutputTrigger и

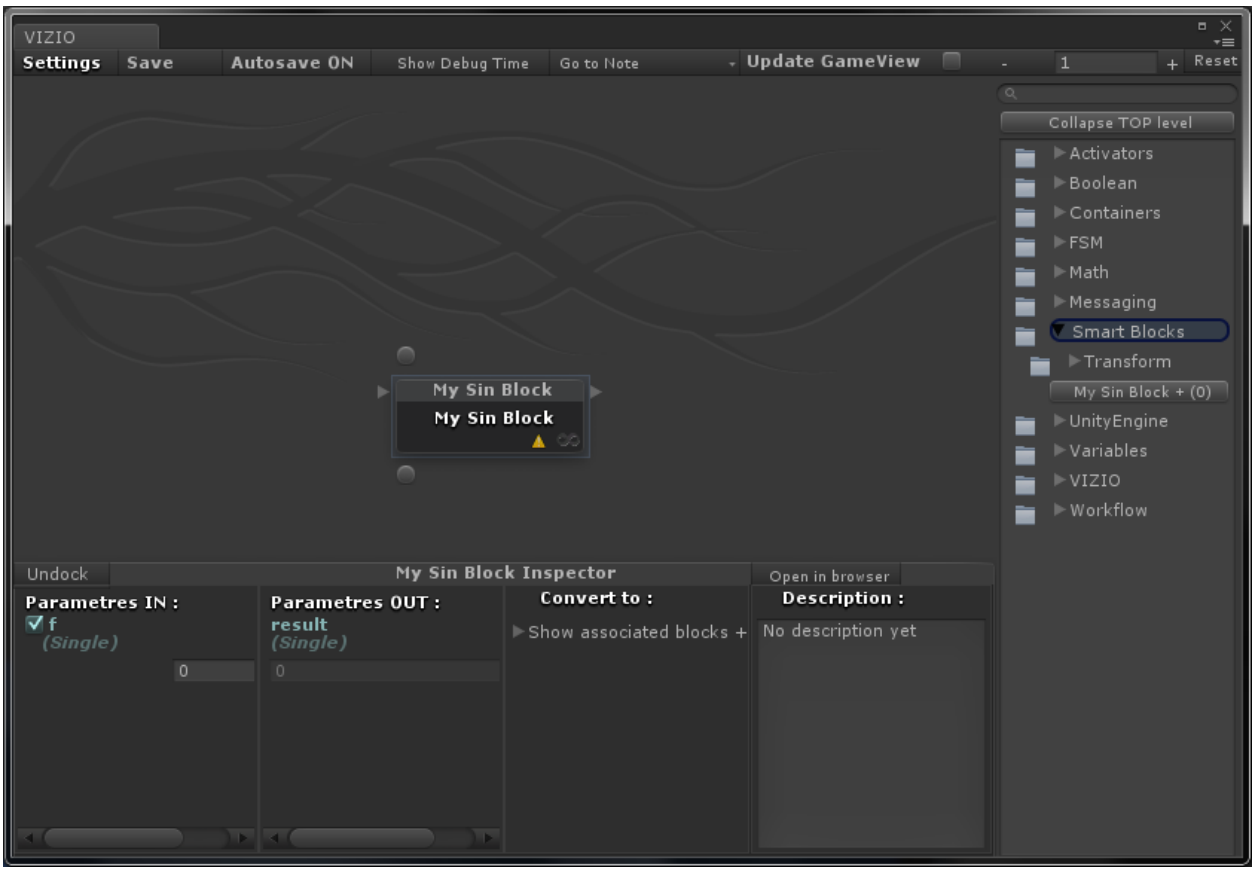
передать ей в качестве параметра – имя выходного триггера. В нашем случае «Exit».

```
[EntryTrigger]
public void In()
{
  result.Value = Mathf.Sin((float)f.Value);

  ActivateTrigger("Exit");
}
```

Входные триггеры – это функции, выполняющие какое-либо действие. Функция входного триггера должна быть публичной (public), иметь тип void и не иметь параметров. Так же, чтобы среда посчитала ее входным триггером – она должна быть помечена атрибутом EntryTrigger.

Наш входной триггер с именем In выполняет расчет функции Mathf.Sin (http://unity3d.com/support/documentation/ScriptReference/Mathf.Sin.html), присваивает результат переменной result и активирует выходной триггер с именем «Exit». Так как у нас всего один триггер в блоке – функцию ActivateTrigger можно так же вызвать без параметров. Это повысит производительность, вызывая первый триггер блока не выполняя поиск триггера по имени.

Разместив данный скрипт в проекте или скомпилировав его в dll и поместив в папку Plugins проекта, Vizio автоматически найдет его и включит в инструментальную панель. Результат создания блока будет следующий:



Создание пользовательского инспектора (Custom Inspector) для логического блока.

Создание пользовательского инспектора для блока позволяет управлять его данными вручную. Пользовательский инспектор выводится в область инспектора Vizio, отведенную под входные и выходные переменные.

Для того чтобы реализовать инспектор для своего блока, необходимо создать класс, унаследованный от класса LogicBlockInspector, пометить его атрибутом CustomInspector (и класс и атрибут находятся в пространстве имен Antares.Vizio.Editor) и в качестве параметра атрибута указать тип класса блока, для которого создается инспектор. Скрипт должен быть помещен в папку Editor проекта (dll в папку Plugins/Editor).

Класс LogicBlockInspector имеет переопределяемое событие OnInspectorGUI, которое выполняет отрисовку инспектора. Свойство класса target имеет тип LogicBlock и имеет ссылку на редактируемый логический блок.

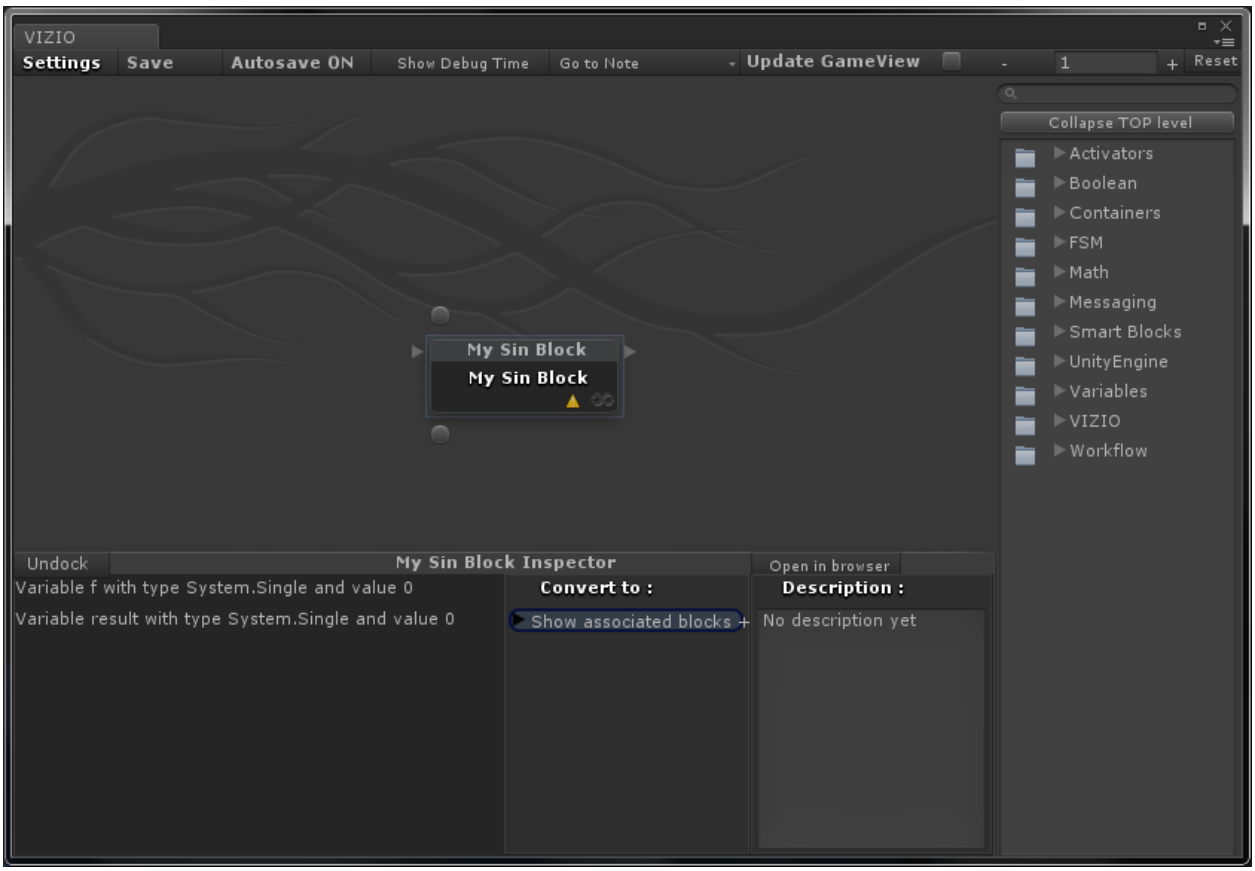
Давайте рассмотрим пример создания простого инспектора для нашего ранее созданного блока.

```
using Antares.Vizio.Editor;
using Antares.Vizio.Runtime;
using UnityEngine;

[CustomInspector(typeof(MySinBlock))]
public class MySinBlockInspector : LogicBlockInspector
{

  public override void OnInspectorGUI()
  {
    foreach (Variable variable in target.variables)
    {
      GUILayout.Label(
        string.Format("Variable {0} with type {1} and value {2}",
          variable.Name, variable.Type.FullName, variable.Value));
    }
  }
}
```

В данном примере объявлен класс MySinBlockInspector, унаследованный от класса LogicBlockInspector. Класс помечен атрибутом CustomInspector, которому в качестве параметра передан тип нашего ранее созданного блока.

В классе перегружено событие OnInspectorGUI, в котором выполняется последовательный вывод данных обо всех переменных блока, полученного через поле target. Все переменные блока содержатся в свойстве LogicBlock.variables.

Результат работы данного инспектора приведен на следующем скриншоте:



Следует заметить, что тип float в Mono или Net Framework является типом System.Single, это можно понять по результатам работы созданного нами инспектора.