WebGL Call C# Method System asset

The **WebGL Call C# Method System** provides a way to call your methods from the browser's native JavaScript in WebGL builds, and obtain the method's response. The system is thus an extension over the standard Unity's *SendMessage* capabilities, which do not let you get the return value or call methods with more than one parameter.

Native Unity SendMessage limitations

Unity can communicate with WebGL through SendMessage (as described in the manual) but SendMessage has, among others, the following limitations:

- Cannot return any value from a method.
- Cannot call methods that take more than a single parameter.

This is inconvenient, because most methods will want to return values, and also because that means that you have to code your unity-side methods explicitly to be able to receive messages (taking a single parameter when you may want to take more, for instance).

Features of the WebGL Call C#

Method System asset

- Can call methods from browser-native JavaScript, and get the return value.
- Can call methods with multiple parameters.
- Full source code provided, with well-written and commented C# code, and with editor-runnable unit tests.

What is included

This package includes several components.

Call C# Method System

Contains the files that are actually needed for the system to work:

- The RECEIVER prefab and internal scripts.
- The native JavaScript script file that should be referenced from the WebGL template.

Call C# Method System Sample

Contains a sample scene that can be used to see how the Call C# Method System is configured and works.

Documentation

The Documentation folder contains this file, which describes the

system and how to use it.

How to setup from scratch

This section will describe step-by-step how to setup the system. Even then, you may find it easier to just have a look at the provided *Call C# Method System Sample Scene*.

1. Setup the RECEIVER object

Add the RECEIVER prefab to your scene. The game object must be called “RECEIVER”. This is necessary for everything to work.

2. Reference the JavaScript script file from your WebGL template

Warning: This step is more complicated than it seems, due to some Unity limitations in how assets are handled.

In your HTML template files, you need to reference the `callmethodsystem.js` script.

You can find that script in `Assets/CallMethodSystem/WebGLTemplates/SampleScene/CallMethodS`
You should *copy* the file into your own template’s folder, for instance, to:
`Assets/WebGLTemplates/YourTemplate/CallMethodSystem/callmethod`

As you can see, the termination (`REMOVE_THIS_TERMINATION`) should also be removed.

Once you have done that, to reference it from the HTML you can, for instance, add to the `<head>` section the following code:

```
    <script src="CallMethodSystem/callmethodsystem.js"></
script>
```

It does not need to be in the `head` section; anywhere else should do as well.

Alternatively, rather than modifying an existing template, you can just use one of the provided ones. You can find the WebGL templates in `Call Method System Sample/WebGLTemplates`. There are two provided templates.

- The Sample one is the one used for the Sample scene, and it shows how the call method system works by calling a few methods from that scene.
- The Empty one is empty, but references the `callmethodsystem.js` JavaScript so it can be extended easily.

To make them available, you need to copy the folder you choose into the “WebGLTemplates” folder within your root assets’s folder. That is, only folders in `Assets/WebGLTemplates` will actually be interpreted by Unity as WebGL templates.

It is also important to *remove the* `REMOVE_THIS_TERMINATION` termination from the `callmethodsystem.js` file name.

For those who are curious, those relatively burdensome steps stem from the facts that:

- Unity does not allow assets to place files outside their own directory. That is, we cannot directly put the sample files into your 'Assets/WebGLTemplates' folder.
- Unity automatically interprets JavaScript files (ending in .js) that are not within the 'Assets/WebGLTemmplates' folder as Unity Script files. (And in this case they are native JavaScript files - Not Unity Script).

Using the Call C# Method System

To call a C# method from your browser's native JavaScript, you just need to use the `callMethod` function. The `callMethod` function is defined in the `callmethodsystem.js` script file, which should be referenced from your HTML page as described in the "Setting up the system" section.

The `callMethod` function is declared as follows:

```
/**
Invokes the specified method on the specified Unity objec
```

```
t, and reports the value through callbacks.

For performance reasons, similarly to Unity's native Send
Message, we do not support method overloading. Thus, the
name
of the method you want to invoke must be unique within ev
ery method linked to the object. If the object has more t
han one
component, then there can be a single method with that na
me for both.

Arguments passed to the method must be strings or numbers
. The method attributes in the C# side must be matched ex
actly.
A string in JavaScript will expect a "string" in C#. A fl
oating-point number in JavaScript will expect a "double"
in C#.
An integer number in JavaScript will expect a "long" in C
#.

@param objectName: Unity name for the object on which we
have a component whose method we want to call.
@param methodName: Name of the method to call.
@param args: [Optional] A list (a JavaScript Array) with
the arguments to call the function with. They must match
the function EXACTLY.
Otherwise an exception will be raised in the C# side.
@param onSuccessCallback: [Optional] JavaScript callback
```

```
that will receive the value that the called method return
s.
@param onErrorCallback: [Optional] JavaScript callback th
at will be notified when the request fails. It may receiv
e a message as an argument, containing the message of the
 original raised exception.
*/
function callMethod(objectName, methodName, args, onSucce
ssCallback, onErrorCallback)
```

As an example, let's say that in our Unity scene we have an object called `Player` we have a C# method called `long ApplyDamage(long dmg, string type)` that applies the specified damage, of the specified type (either 'fire' or 'ice' or 'normal') and returns the resulting "health" of our character after applying it. To call it from JavaScript, we would do:

```
callMethod("Player", "ApplyDamage", [100, "fire"],
    function(health) {
        console.log("Our new health is: ", health);
    }, function(errorMessage) {
        console.error("Our attempt to ApplyDamage failed
with message: ", errorMessage);
    });
```

Though this might seem like quite a lot of code, please note that most of it are actually just the result callbacks. The first function that is

passed as a callback will receive the return value of the function we called if the call is successful. The second, will receive the error message if the call **is not** successful. All callbacks are optional, so if we trust that it will not fail, we could just omit the error callback. And if we don’t care about the return value, we can omit the success callback as well. So we could do, for instance:

```
callMethod("Player", "ApplyDamage", [100, "fire"]);
```

Please note that something very important is that the method you are trying to call actually exists, and that the arguments you are passing match the parameters that the method expects **exactly**. C# is strong-typed, so the type must be quite specific. The list of accepted types are the following:

- JavaScript number with no decimals (e.g., 2) is compatible with a C# long or a C# double. It is **not** compatible with a C# int or float.
- JavaScript number with decimals (e.g., 2.2) is compatible with a C# double. It is **not** compatible with a C# float.
- JavaScript string is compatible with a C# string. Even if it contains something that looks like a number, it is only compatible with a C# string.
- Objects, lists and arrays are not supported. Please, see the FAQ section for a common work-around to this need.

Just as an example, with the hypothethical aforesaid ApplyDamage function, doing the following would result in an error:

```
callMethod("Player", "ApplyDamage", [100.2, "fire"]),
    function(health) {
        console.log("This will never be called.");
    }, function(errorMessage) {
        console.error("This will result in a parameter-mi
smatch error.")
    });
```

The cause is, of course, that the C# function expects a ‘long’ and is being passed a number with decimals, which would require a ‘double’.

Questions & Answers

Why do I get an argument mismatch exception when I call my C# method?

It is important to ensure that you are calling the method with arguments of the right type. Take into account that C# is strong-typed, so JavaScript variables will automatically be converted to C# types, which will need to match. A number with no decimals (e.g., “2”) in JavaScript will be compatible with either a long or a double in C#. A number with decimals (e.g., “2.2”) in JavaScript will be compatible with only a double in C#. A string in JavaScript will be compatible with a string in C#. A boolean with a boolean. And other types are unsupported. Particularly, objects, lists and dictionaries are unsupported.

What can I do if my C# method takes an object/list/dictionary as an argument?

Unfortunately, you cannot call such a method as-is from JavaScript. A work-around is to create a wrapper method in the C# side which either accepts primitive types, or which accepts a string as a parameter. Then, from JavaScript, you can encode your parameters into JSON (through for example `JSON.stringify([2, 5, 1, 3])`), and then decode that string in the C# method wrapper, and then call the actual C# method.

Can I get the return value 'normally', without a callback?

No. There is currently no way to do something such as `myVal = callMethod('SCRIPT', 'SumValue', [2, 3])`. Internally, we rely on Unity's SendMessage, which is asynchronous; and JavaScript itself is inherently asynchronous, so we cannot do much at that respect. The bright side is that asynchronous code will tend to be faster than waiting probably would (if it were possible).

I am calling a C# method just after loading the page and I am getting an

exception, what can I do?

The Unity engine and applications take time to load, and until they do, calling methods is likely to fail. You need to wait for the loading process to complete before calling a method.

I cannot choose my WebGL template when building. What can I do?

Only the folders in `Assets/WebGLTemplates` are interpreted as WebGL templates. It's not enough to simply call the folder "WebGLTemplates". Therefore, the `index.html` file of your template should be in a path such as: `Assets/WebGLTemplates/MyTemplateFolder/index.html`.

I am getting a JavaScript "callMethod is undefined" error.

You have probably not renamed the `callmethodsystem.js.REMOVE_THIS_TERMINATION` to `callmethodsystem.js`, or you have not referenced the `callmethodsystem.js` script from your WebGL template, or Chrome is not finding it properly. Please, ensure that the `callmethodsystem.js` has been added and is being loaded properly. You can check through the Chrome Developer Tools. For more information, see the Setup section of this document.

How can I ask other questions, get support, provide feedback or make suggestions?

Please, send an e-mail to luis.rodgi@gmail.com.

Version history

Version 1.0 (14 November 2016)

- Initial release version.

Acknowledgements

- To those of you who have bought this package.
- To those of you who have provided very useful feedback.

Contact

This package has been developed by Luis Rodríguez. You can contact him to send your feedback, questions or suggestions at luis.rodgi@gmail.com.