
# VRC OWO System World Integration.

In this documentation we will teach you the Advanced Sensation details of the OWO World Integration (In this documentation we use O.W.I as an abbreviation of the OWO World Integration name to make it easier to read and less repetitive)

This includes:

- Single Sensations
- Multi Sensations
- Setting Muscles
- What each piece code does and how to use it.
  (We do not cover every single use case, this only explains how to code the basics)

(If you have not gone over the Basic Setup please see the documentation on our Github page or in the /Assets/O.W.I/Documents folder in your project after importing the VRC_O.W.I.unitypackage)

Requirements:

A VRChat World Project

Have the VRC_O.W.I.unitypackage Installed

(The GitHub readme.txt file is included in the files under the path: /Assets/O.W.I/Documents)

(within this documentation we mention the GitHub readme.txt file, this is where you can find premade code to help you along if you get stuck.)

# VRC OWO System World integration.

```
// Muscle Collider Names
private readonly string pectoralL = "owo_suit_Pectoral_L";
private readonly string pectoralR = "owo_suit_Pectoral_R";
private readonly string dorsalL = "owo_suit_Dorsal_L";
private readonly string dorsalR = "owo_suit_Dorsal_R";
private readonly string armL = "owo_suit_Arm_L";
private readonly string armR = "owo_suit_Arm_R";
private readonly string lumbarL = "owo_suit_Lumbar_L";
private readonly string lumbarR = "owo_suit_Lumbar_R";
private readonly string abdominalL = "owo_suit_Abdominal_L";
private readonly string abdominalR = "owo_suit_Abdominal_R";
```

```
// Muscle String Names
private readonly string pectoralLm = "\"pectoral_L\": 100";
private readonly string pectoralRm = "\"pectoral_R\": 100";
private readonly string dorsalLm = "\"dorsal_L\": 100";
private readonly string dorsalRm = "\"dorsal_R\": 100";
private readonly string armLm = "\"arm_L\": 100";
private readonly string armRm = "\"arm_R\": 100";
private readonly string lumbarLm = "\"lumbar_L\": 100";
private readonly string lumbarRm = "\"lumbar_R\": 100";
private readonly string abdominalLm = "\"abdominal_L\": 100";
private readonly string abdominalRm = "\"abdominal_R\": 100";
```

# Single Sensation:

( For after you already have a VRChat Project and Installed the O.W.I package)

To start coding your own sensations, we have to get an understanding of how the O.W.I system works.

## Muscle Collider Names:

The muscle collider names are used to check against the “O.W.I Avatar objects.Prefab”. We need these strings to check for a “on collision” or an “on trigger” event on any of the corresponding muscle objects.

This allows us to know what Muscle(s) to target for the following sensation later on in the script.

## Muscle String Names:

The muscle string names are used in the output string to tell the external client what muscle(s) we are trying to trigger.

The numbers after the naming of each Muscle is the Override control, this can either be static (as shown in the image) if we don't care about the intensity on each muscle.

We can set each of these individually so that each muscle had a percentage of the overall intensity we set later on in the script.

This can be done Dynamically through editable variables in the inspector.
( Keep in mind you'll need to add a public variable so that we can set it through the inspector instead of opening up our script every time you need to change it.)

# VRC OWO System World integration.


```
// Sensation String Parts
private string start = "VRC_OWO_WorldIntegration:[{\"priority\": ";
private readonly string end = "}}]";
```

## Single Sensation:

String Parts:
The following string variables are for use later on in the code when we are putting together our string that we send into the Console.log function for the external client to use and parse the data for the OWO system to use. ( please remember to set theses up ahead of time. )

Start:
This is the start of our string that will be send into the log  this holds our identifier as well as the start of our string. We use an identifier so that the external client doesn't have to scan  everything in the log before parsing any data, this way it makes our system faster.
The identifier  can be seen in the image on the left “VRC_OWO_WorldIntigration:”.

We also need to make sure that your string and our string are formatted the same, if not the external client will ignore the data sent as it will deem it to be corrupt.

Entire start string: “VRC_OWO_worldIntegration: [{\”priority\”: “

End:
This is just the ending section of our string, this is used to close of the data and make sure the external client can parse it correctly.

Entire string: “}}]”

(please feel free to copy paste this information from the GitHub Readme.txt file)

# VRC OWO System World integration.

```
[Header("Sensation Settings")]
[SerializeField, Tooltip("Value Decides if it interrupts the previous sensation")]
private int sensationPriority = 1;
[SerializeField, Tooltip("Name for the Sensation event.")]
private string sensationName = "Default Name";
[SerializeField, Tooltip("Frequency for the Sensation event.")]
[Range(1, 100)]
private int frequency = 100;
[SerializeField, Tooltip("Duration of the Sensation event.")]
[Range(0.1f, 20)]
private float duration = 0.1f;
```

# Single Sensation:

## Sensation Settings:

As we talked about in the last section we may need to set variables after we are done making the script, this is where out Sensation settings reside.
This is also where you can add anymore editable variables you need for your code.
(keep in mind that the order you have the variables in is the order they will show up in the inspector.)

As you can see most of the variables here have a range, the following will explain each variable and why their ranges are set how they are.

## Sensation priority:

The value put into this variable decides if it interrupts the previous sensation. The higher the number the higher the priority.

A priority of 3 will stop a sensation that is lower then its own. E.g: if we have a sensation of priority 2 playing and a sensation of priority 3 that comes in, we will play the priority 3 sensation and completely stop the sensation before it.

## Sensation name:

This is not used anywhere in the calculations of the code, it is simply so we know what sensation is being played, it's mainly for telling each sensation apart.

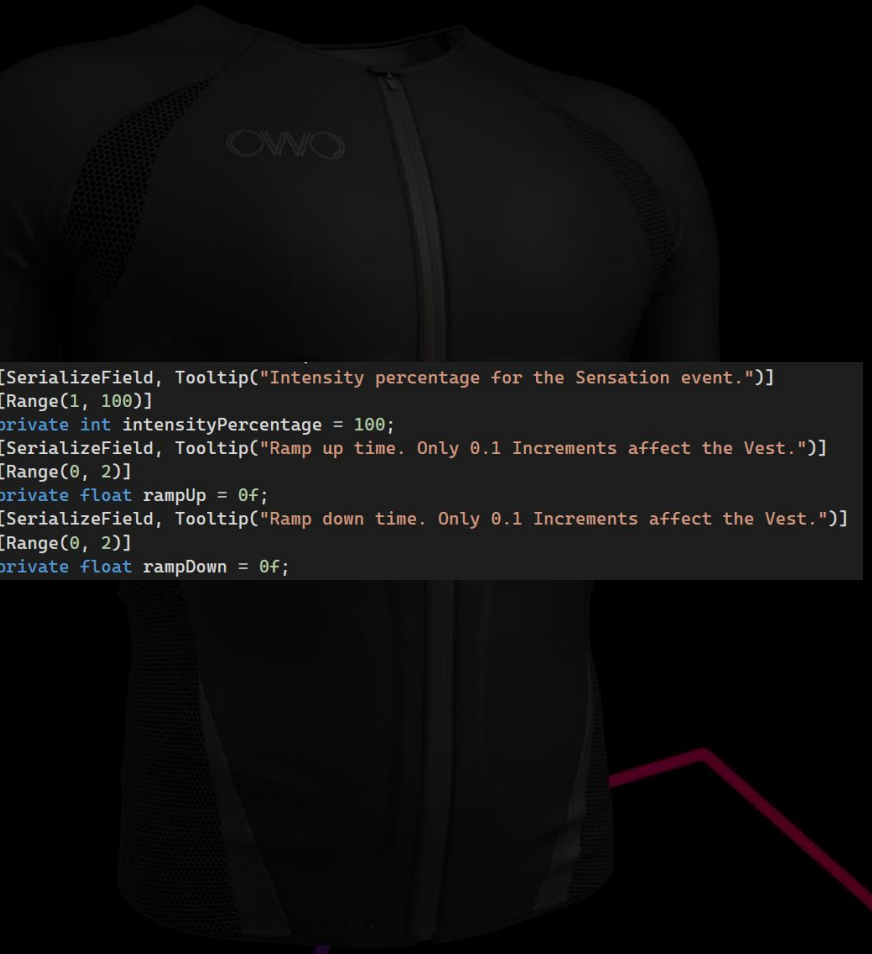
## Frequency:

This is the pulses of the electrical current being sent through the person's muscles.
This has a range of 1 - 100 as this is the min and max that the OWO system can supply.

## Duration:

This is the length of sensation that we send to the OWO system, This is done is seconds.
This has a range is 0.1 - 20 as this is the limitation of the OWO system.

# VRC OWO System World integration.



```
[SerializeField, Tooltip("Intensity percentage for the Sensation event.")]
[Range(1, 100)]
private int intensityPercentage = 100;
[SerializeField, Tooltip("Ramp up time. Only 0.1 Increments affect the Vest.")]
[Range(0, 2)]
private float rampUp = 0f;
[SerializeField, Tooltip("Ramp down time. Only 0.1 Increments affect the Vest.")]
[Range(0, 2)]
private float rampDown = 0f;
```

# Single Sensation:

## Sensation Settings:

Continuing from the last page the following will explain each variable and why their ranges are set how they are.

## Intensity Percentage:

This is the percentage of the users max config, OWO system side. This variable controls how strong the sensation feels.
Its range is 1 - 100
To understand how the maths of this works.

If the users max config is 30% and our Intensity is 50% the overall intensity that the user will feel is 15%.
Another example being:
The users max config is 45% and our intensity is 67% the overall intensity that the user feels is 30%

## Ramp Up:

This controls the time it takes for the intensity to hit its set value from the start of the sensation.
Its range is 0 - 2 this is done in seconds. (you can use decimals this only works in increments of 0.1)

## Ramp Down:

This controls the time is takes for the intensity to reach 0 at the end of the sensation.
Its range is 0 - 2 this is done in seconds (you can use decimals this only works in increments of 0.1)

E.g: if we have a duration of 2 seconds with a ramp down of 0.5 seconds then the intensity will only start to go down once we hit, 1.5 seconds so that it reaches 0 at the end of the sensation.

# VRC OWO System World integration.

## Setting Muscles:

```
private void OnTriggerEnter(Collider other)
{
    string currentMuscle = "";

    switch (other.name)
    {
        case pectoralL:
            currentMuscle += pectoralLm;
            muscleTriggered = true;
            break;
        case pectoralR:
            currentMuscle += pectoralRm;
            muscleTriggered = true;
            break;
        case dorsalL:
            currentMuscle += dorsalLm;
            muscleTriggered = true;
            break;
        case dorsalR:
            currentMuscle += dorsalRm;
            muscleTriggered = true;
            break;
        case armL:
            currentMuscle += armLm;
            muscleTriggered = true;
            break;
        case armR:
            currentMuscle += armRm;
            muscleTriggered = true;
            break;
        case lumbarL:
            currentMuscle += lumbarLm;
            muscleTriggered = true;
            break;
        case lumbarR:
            currentMuscle += lumbarRm;
            muscleTriggered = true;
            break;
        case abdominalL:
            currentMuscle += abdominalLm;
            muscleTriggered = true;
            break;
        case abdominalR:
            currentMuscle += abdominalRm;
            muscleTriggered = true;
            break;
        default:
            return;
    }
    if (muscleTriggered)
    {
        if (triggerCount <= 1)
        {
            triggerCount++;
        }
        shouldProcess = true;
    }
    if (triggerCount == 1)
    {
        triggeredMuscles = currentMuscle;
    }
    muscleTriggered = false;
}
```

So this is where things get tricky, there are two options when picking how we want the sensation to play out.

First option: (Image on the left)
Playing the sensation through the muscles that was hit.
(if you want the first option please go to page 7 and skip page 8)

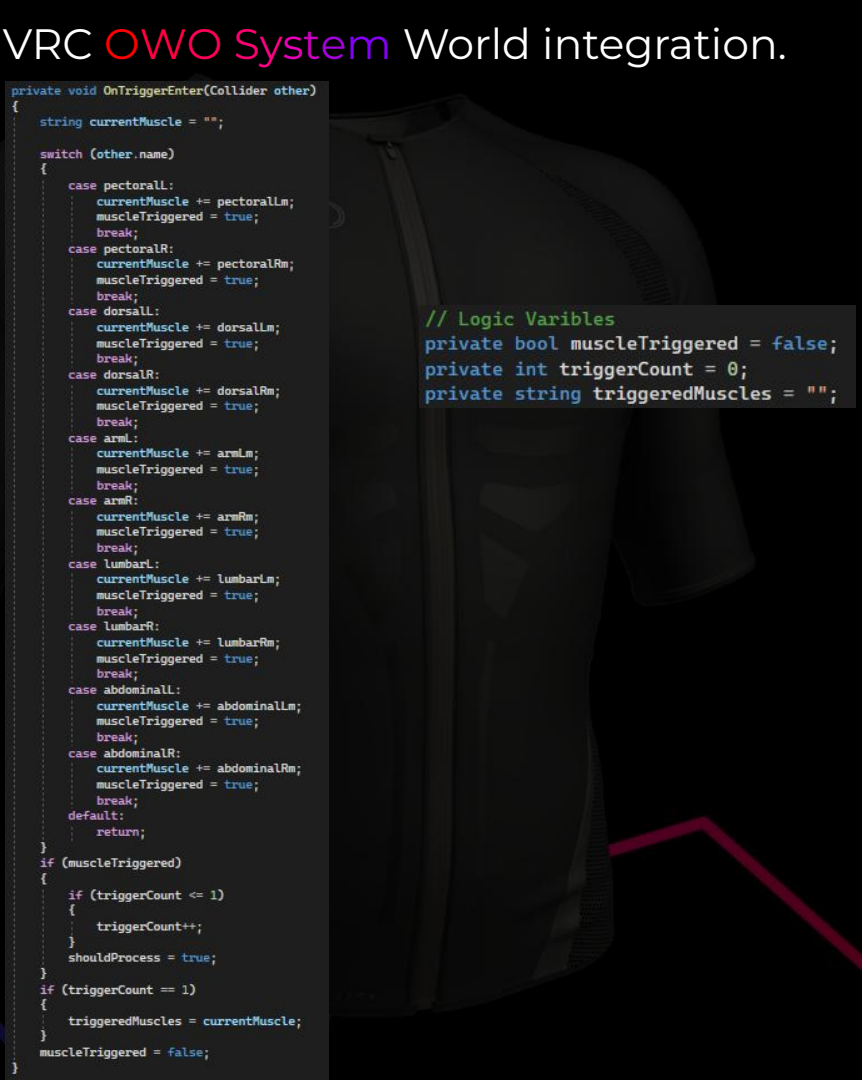
Second option: (Image on the right)
Playing the sensation through muscles that are predefined in the inspector before building the map.( The Sensation will play on all the muscles at the same time)
(if you want the first option please go to page 8)

```
private void Start()
{
        if (UseMusclePectoralLeft)
        {
            builtMuscles += pectoralLm + ",";
        }
        if (UseMusclePectoralRight)
        {
            builtMuscles += pectoralRm + ",";
        }
        if (UseMuscleDorsalLeft)
        {
            builtMuscles += dorsalLm + ",";
        }
        if (UseMuscleDorsalRight)
        {
            builtMuscles += dorsalRm + ",";
        }
        if (UseMuscleArmLeft)
        {
            builtMuscles += armLm + ",";
        }
        if (UseMuscleArmRight)
        {
            builtMuscles += armRm + ",";
        }
        if (UseMuscleLumbarLeft)
        {
            builtMuscles += lumbarLm + ",";
        }
        if (UseMuscleLumbarRight)
        {
            builtMuscles += lumbarRm + ",";
        }
        if (UseMuscleAbdominalLeft)
        {
            builtMuscles += abdominalLm + ",";
        }
        if (UseMuscleAbdominalRight)
        {
            builtMuscles += abdominalRm;
        }
        triggeredMuscles = builtMuscles.TrimEnd(',');
}
```

# VRC OWO System World integration.

```
private void OnTriggerEnter(Collider other)
{
    string currentMuscle = "";

    switch (other.name)
    {
        case pectoralL:
            currentMuscle += pectoralLm;
            muscleTriggered = true;
            break;
        case pectoralR:
            currentMuscle += pectoralRm;
            muscleTriggered = true;
            break;
        case dorsalL:
            currentMuscle += dorsalLm;
            muscleTriggered = true;
            break;
        case dorsalR:
            currentMuscle += dorsalRm;
            muscleTriggered = true;
            break;
        case armL:
            currentMuscle += armLm;
            muscleTriggered = true;
            break;
        case armR:
            currentMuscle += armRm;
            muscleTriggered = true;
            break;
        case lumbarL:
            currentMuscle += lumbarLm;
            muscleTriggered = true;
            break;
        case lumbarR:
            currentMuscle += lumbarRm;
            muscleTriggered = true;
            break;
        case abdominalL:
            currentMuscle += abdominalLm;
            muscleTriggered = true;
            break;
        case abdominalR:
            currentMuscle += abdominalRm;
            muscleTriggered = true;
            break;
        default:
            return;
    }
    if (muscleTriggered)
    {
        if (triggerCount <= 1)
        {
            triggerCount++;
        }
        shouldProcess = true;
    }
    if (triggerCount == 1)
    {
        triggeredMuscles = currentMuscle;
    }
    muscleTriggered = false;
}
```



```
// Logic Varibles
private bool muscleTriggered = false;
private int triggerCount = 0;
private string triggeredMuscles = "";
```

## Setting Muscles in order:

This is the code for how we will set the muscle that was hit.
However we need to set some more variables before we can add this.
The following explains what variables you need to add.

**MuscleTriggered:**
This is the logic to advance the count when we actually touch a muscle.

**TriggerCount:**
This is used to ignore muscles hit after the first.

**TriggeredMuscles:**
This is the first muscle that was hit, we set the name of the first muscle that was hit inside the variable.

This code works by using the “OnTriggerEnter” event,
When this event is triggered we check what was hit, by checking if the name of the object was the same as one of our muscle collider string names. This is done because if there are other triggers in the world that is entered by the player, that will also set off the “OnTriggerEnter” event.
Once this is checked, we add the muscle that was hit to our corresponding “triggeredMuscle” variable.

# VRC OWO System World integration.

## Setting Muscles Predefined:

```
private void Start()
{
        if (UseMusclePectoralLeft)
        {
            builtMuscles += pectoralLm + ",";
        }
        if (UseMusclePectoralRight)
        {
            builtMuscles += pectoralRm + ",";
        }
        if (UseMuscleDorsalLeft)
        {
            builtMuscles += dorsalLm + ",";
        }
        if (UseMuscleDorsalRight)
        {
            builtMuscles += dorsalRm + ",";
        }
        if (UseMuscleArmLeft)
        {
            builtMuscles += armLm + ",";
        }
        if (UseMuscleArmRight)
        {
            builtMuscles += armRm + ",";
        }
        if (UseMuscleLumbarLeft)
        {
            builtMuscles += lumbarLm + ",";
        }
        if (UseMuscleLumbarRight)
        {
            builtMuscles += lumbarRm + ",";
        }
        if (UseMuscleAbdominalLeft)
        {
            builtMuscles += abdominalLm + ",";
        }
        if (UseMuscleAbdominalRight)
        {
            builtMuscles += abdominalRm;
        }
        builtMuscles = builtMuscles.TrimEnd(',');
}
```

```
private string builtMuscles = "";
[SerializeField] private bool UseMusclePectoralLeft;
[SerializeField] private bool UseMusclePectoralRight;
[SerializeField] private bool UseMuscleDorsalLeft;
[SerializeField] private bool UseMuscleDorsalRight;
[SerializeField] private bool UseMuscleArmLeft;
[SerializeField] private bool UseMuscleArmRight;
[SerializeField] private bool UseMuscleLumbarLeft;
[SerializeField] private bool UseMuscleLumbarRight;
[SerializeField] private bool UseMuscleAbdominalLeft;
[SerializeField] private bool UseMuscleAbdominalRight;
```

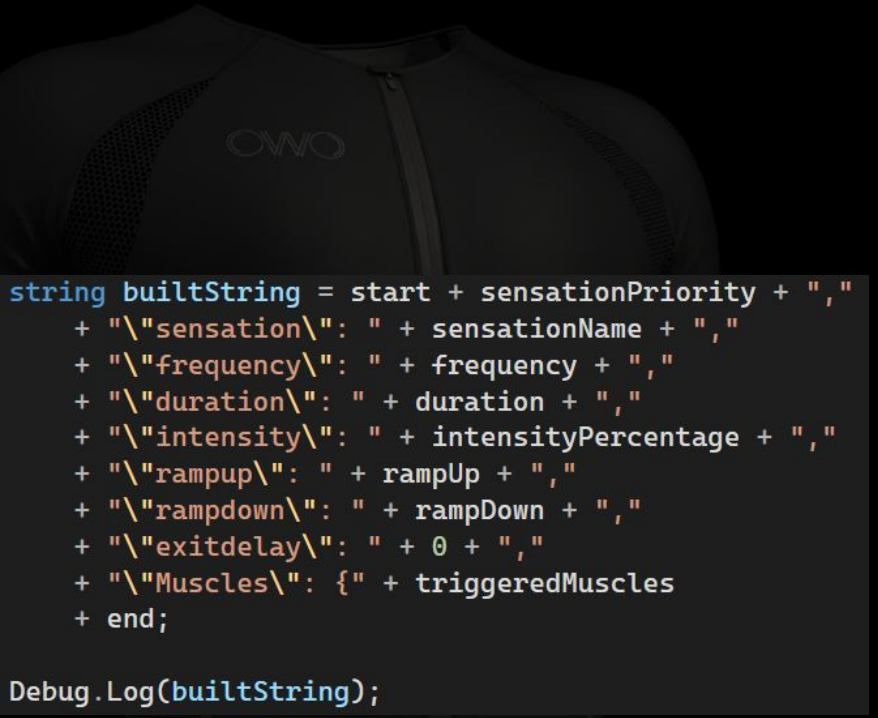
This is the code for how we will set predefined muscles.
However we need to set some more variables before we can add this. The following explains what variables you need to add.

**BuiltMuscles:**
This holds the muscle string names that we set through the code ( see images on the left)

**Use Muscle"Specific Muscle":**
This is a boolean that we set in the inspector this indicates if you are adding the muscle to the "BuiltMuscles" variable or not.

This code works by using the "Start" event.
When the world loads it will check which muscles were selected using the Use Muscle variables and adds them to the "BuiltMuscles" variable for use in the sensation.

# VRC OWO System World integration.



```
string builtString = start + sensationPriority + ","
    + "\"sensation\": " + sensationName + ","
    + "\"frequency\": " + frequency + ","
    + "\"duration\": " + duration + ","
    + "\"intensity\": " + intensityPercentage + ","
    + "\"rampup\": " + rampUp + ","
    + "\"rampdown\": " + rampDown + ","
    + "\"exitdelay\": " + 0 + ","
    + "\"Muscles\": {" + triggeredMuscles
    + end;

Debug.Log(builtString);
```

# Single Sensation Building String:

Building the string data:
Before we can send anything to the console.log function so that the external client can pick it up and use it, we need to make sure that the string is in the correct format.

We started this process earlier on the in the documentation (page 3 string parts:)

We now need to fill out our string in the correct order to make sure that it is formatted correctly.
(please follow the code on the left hand side.)
As seen to the left we need to start with our sensation name, then our frequency, then duration and so on.
(please feel free to copy paste this information from the GitHub Readme.txt file).

If you have used different names for your variables we need to replace them with your variables.
To do this where it say sensationName, Frequency, duration, intensityPercentage, rampUp, RampDown, and triggeredMucles, we need to change each one out to your corresponding variable name without removing any of the spacing or code from that section.

Exit delay is always set to 0 because we are working with a single sensation, so we do not need an Exit delay.

Once this is done, you can test your string and check your console to see if your string matches the format seen below.
(if it doesn't you have done something wrong. Please go back and check your code and compare it to our documentation.)

VRC_OWO_WorldIntegration:[{"priority": 1,"sensation": "Default Name","frequency": 100,"duration": 1,"intensity": 100,"rampup":0,"rampdown":0,"exitdelay":0,"Muscles": {"lumbar_L": 100}}]

VRC OWO System World integration.


# Multi-Sensation Script Examples.

# VRC OWO System World integration.

```
// Sensation Settings
[Header("First Zone Triggered Event")]
[Header("Sensation Building Settings")]
[SerializeField, Tooltip("Value Decides if it interrupts the previous sensation")]
private int sensationPriority = 1;
[SerializeField, Tooltip("Name for the Sensation event.")]
private string sensationName = "Sword Stab";
[SerializeField, Tooltip("Frequency for the Sensation event.")]
[Range(1, 100)]
private int frequency = 60;
[SerializeField, Tooltip("Duration of the Sensation event.")]
[Range(0.1f, 20)]
private float duration = 0.3f;
[SerializeField, Tooltip("Intensity percentage for the Sensation event.")]
[Range(1, 100)]
private int intensityPercentage = 100;
[SerializeField, Tooltip("Ramp up time Where 0.1 = 100ms Only 0.1 Increments affect the Vest.")]
[Range(0, 2)]
private float rampUp = 0f;
[SerializeField, Tooltip("Ramp down time Where 0.1 = 100ms Only 0.1 Increments affect the Vest.")]
[Range(0, 2)]
private float rampDown = 0f;
[SerializeField, Tooltip("Exit delay for the  Sensation event.")]
[Range(0, 20)]
private float exitDelay = 0f;

[Header("Secondary Zone Triggered Event")]
[SerializeField, Tooltip("Name for the Sensation event.")]
private string sensationName2 = "Sword Stab Through";
[SerializeField, Tooltip("Frequency for the Sensation event.")]
[Range(1, 100)]
private int frequency2 = 50;
[SerializeField, Tooltip("Duration of the  Sensation event.")]
[Range(0.1f, 20)]
private float duration2 = 0.3f;
[SerializeField, Tooltip("Intensity percentage for the  Sensation event.")]
[Range(1, 100)]
private int intensityPercentage2 = 80;
[SerializeField, Tooltip("Ramp up time Where 0.1 = 100ms Only 0.1 Increments affect the Vest.")]
[Range(0, 2)]
private float rampUp2 = 0f;
[SerializeField, Tooltip("Ramp down time Where 0.1 = 100ms Only 0.1 Increments affect the Vest.")]
[Range(0, 2)]
private float rampDown2 = 0f;
[SerializeField, Tooltip("Exit delay for the Sensation event.")]
[Range(0, 20)]
private float exitDelay2 = 0f;
```

## Multi-Sensations:

(For use linking two or more individual sensations together.)

### Sensation settings:

The following sensation settings are the same as the single sensation apart from one extra variable.

### Exit Delay:

This controls the amount of time in seconds, before the next sensation plays.(within the same multi-Sensation script)
Its range is 0 - 20 in seconds.
(this was not included in the single sensation as there was no follow up sensation to go after it. Therefore we didn't need it in that script)

The amount of sensations per Multi-Sensation is not limited, HOWEVER for every new sensation, you need a new set for variables, this can be done but ether adding a number on the end of your variable name. Or (if you like to make your life hard) changing the name completely.

This is done because we cannot use the same variable for each sensation because each sensation has different settings. If they were all the same variable they would all have the same settings.

# VRC OWO System World integration.

```
[Header("Sensation Settings")]
[SerializeField, Tooltip("Value Decides if it interrupts the previous sensation")]
private int sensationPriority = 1;
[SerializeField, Tooltip("Name for the Sensation event.")]
private string sensationName = "Default Name";
[SerializeField, Tooltip("Frequency for the Sensation event.")]
[Range(1, 100)]
private int frequency = 100;
[SerializeField, Tooltip("Duration of the Sensation event.")]
[Range(0.1f, 20)]
private float duration = 0.1f;
```

## Multi-Sensations:

### Sensation settings:

### Sensation priority:
The value put into this variable decides if it interrupts the previous sensation. The higher the number the higher the priority.

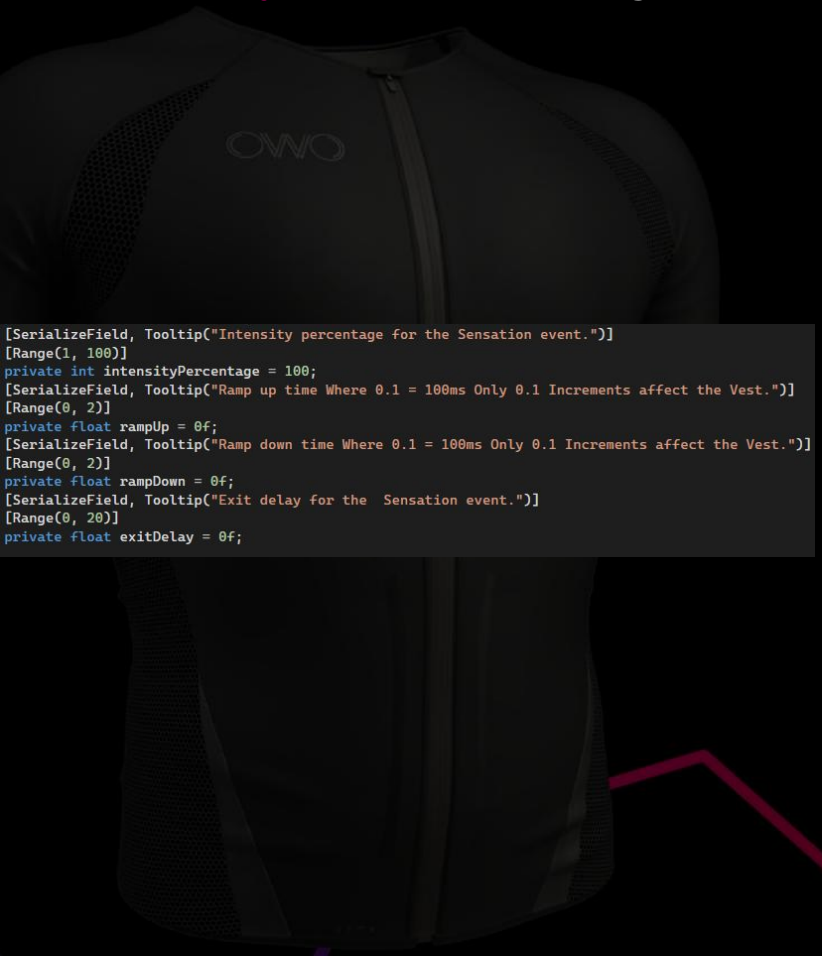
A priority of 3 will stop a sensation that is lower then its own. E.g: if we have a sensation of priority 2 playing and a sensation of priority 3 that comes in, we will play the priority 3 sensation and completely stop the sensation before it.

### Sensation name:
This is not used anywhere in the calculations of the code, it is simply so we know what sensation is being played, it's mainly for telling each sensation apart.

### Frequency:
This is the pulses of the electrical current being sent through the person's muscles. This has a range of 1 - 100 as this is the min and max that the OWO system can supply.

### Duration:
This is the length of sensation that we send to the OWO system, This is done is seconds. This has a range is 0.1 - 20 as this is the limitation of the OWO system.

# VRC OWO System World integration.



```
[SerializeField, Tooltip("Intensity percentage for the Sensation event.")]
[Range(1, 100)]
private int intensityPercentage = 100;
[SerializeField, Tooltip("Ramp up time Where 0.1 = 100ms Only 0.1 Increments affect the Vest.")]
[Range(0, 2)]
private float rampUp = 0f;
[SerializeField, Tooltip("Ramp down time Where 0.1 = 100ms Only 0.1 Increments affect the Vest.")]
[Range(0, 2)]
private float rampDown = 0f;
[SerializeField, Tooltip("Exit delay for the  Sensation event.")]
[Range(0, 20)]
private float exitDelay = 0f;
```

# Multi-Sensations:

## Sensation settings:

### Intensity Percentage:

This is the percentage of the users max config, OWO system side. This variable controls how strong the sensation feels.
Its range is 1 - 100
To understand how the maths of this works.

If the users max config is 30% and our Intensity is 50% the overall intensity that the user will feel is 15%.
Another example being:
The users max config is 45% and our intensity is 67% the overall intensity that the user feels is 30%

### Ramp Up:

This controls the time it takes for the intensity to hit its set  value from the start of the sensation.
Its range is 0 - 2 this is done in seconds. (you can use decimals this only works in increments of 0.1)

### Ramp Down:

This controls the time is takes for the intensity to reach 0 at the end of the sensation.
Its range is 0 - 2  this is done in seconds (you can use decimals this only works in increments of 0.1)

### Exit Delay:

This controls the amount of time in seconds, before the next sensation plays.(within the same multi-Sensation script)
Its range is 0 - 20 in seconds.
(this was not included in the single sensation as there was no follow up sensation to go after it. Therefore we didn't need it in that script)

# VRC OWO System World integration.

```
// String Parts
private readonly string start = "VRC_OWO_WorldIntegration:[{ \"priority\":";
private readonly string sensationNameStart = "\"sensation\": \"";
private readonly string separator = "}},{";
private readonly string end = "}}]";
```

## Multi-Sensations:

### String Parts:

The following string parts are the same as the single sensation string parts. With the addition of sensation name start and separator they will be explained below.
(please review page 3 for the information on “Start” and “End” see image on the left for how the code should look.) ( please remember to set these up ahead of time. )

### Sensation Name Start:

This is used to tell us what sensation is being played during the playthrough of the Multi-Sensation, this helps us understand what is going to out external client and if it has the correct formatting.
(This name is not used in any of the maths apart from building the strings)

### Separator:

This is used to separate each single sensation that makes up the Multi-Sensation.
(yes that's it)

(please feel free to copy paste this information from the GitHub Readme.txt file)

# VRC OWO System World integration.

```
// Muscle Collider Names
private readonly string pectoralL = "owo_suit_Pectoral_L";
private readonly string pectoralR = "owo_suit_Pectoral_R";
private readonly string dorsalL = "owo_suit_Dorsal_L";
private readonly string dorsalR = "owo_suit_Dorsal_R";
private readonly string armL = "owo_suit_Arm_L";
private readonly string armR = "owo_suit_Arm_R";
private readonly string lumbarL = "owo_suit_Lumbar_L";
private readonly string lumbarR = "owo_suit_Lumbar_R";
private readonly string abdominalL = "owo_suit_Abdominal_L";
private readonly string abdominalR = "owo_suit_Abdominal_R";
```

```
// Muscle String Names
private readonly string pectoralLm = "\"pectoral_L\": 100";
private readonly string pectoralRm = "\"pectoral_R\": 100";
private readonly string dorsalLm = "\"dorsal_L\": 100";
private readonly string dorsalRm = "\"dorsal_R\": 100";
private readonly string armLm = "\"arm_L\": 100";
private readonly string armRm = "\"arm_R\": 100";
private readonly string lumbarLm = "\"lumbar_L\": 100";
private readonly string lumbarRm = "\"lumbar_R\": 100";
private readonly string abdominalLm = "\"abdominal_L\": 100";
private readonly string abdominalRm = "\"abdominal_R\": 100";
```

## Setting Muscles:

This is the same as how we set them up on the single sensation.

### Muscle Collider Names:

The muscle collider names are used to check against the “O.W.I Avatar objects.Prefab”. We need these strings to check for a “on collision” or an “on trigger” event on any of the corresponding muscle objects.

This allows us to know what Muscle(s) to target for the following sensation later on in the script.

### Muscle String Names:

The muscle string names are used in the output string to tell the external client what muscle(s) we are trying to trigger.

The numbers after the naming of each Muscle is the Override control, this can either be static (as shown in the image) if we don't care about the intensity on each muscle.

We can set each of these individually so that each muscle had a percentage of the overall intensity we set later on in the script.

This can be done Dynamically through editable variables in the inspector.
( Keep in mind you'll need to add a public variable so that we can set it through the inspector instead of opening up our script every time you need to change it.)

# Setting Muscles:

```
private void OnTriggerEnter(Collider other)
{
    string currentMuscle = "";

    switch (other.name)
    {
        case pectoralL:
            currentMuscle += pectoralLm;
            break;
        case pectoralR:
            currentMuscle += pectoralRm;
            break;
        case dorsalL:
            currentMuscle += dorsalLm;
            break;
        case dorsalR:
            currentMuscle += dorsalRm;
            break;
        case armL:
            currentMuscle += armLm;
            break;
        case armR:
            currentMuscle += armRm;
            break;
        case lumbarL:
            currentMuscle += lumbarLm;
            break;
        case lumbarR:
            currentMuscle += lumbarRm;
            break;
        case abdominalL:
            currentMuscle += abdominalLm;
            break;
        case abdominalR:
            currentMuscle += abdominalRm;
            break;
        default:
            return;
    }

    muscleTriggered = true;

    if (muscleTriggered)
    {
        if (triggerCount <= 1)
        {
            triggerCount++;
        }
        shouldProcess = true;
    }
    if (triggerCount == 1)
    {
        triggeredMuscles = currentMuscle;
    }
    else if (triggerCount == 2)
    {
        triggeredMuscles2 = currentMuscle;
    }
    muscleTriggered = false;
}
```

## Picking how we detect the muscles:

So this is where things get tricky, there are two options when picking how we want the sensation to play out.

First option: (Image on the left)
Playing the sensation through the muscles in the order that they were hit.
(if you want the first option please go to page 17)

Second option: (Image on the right)
Playing the sensation through muscles that are predefined in the inspector before building the map.( The Sensation will play on all the muscles at the same time)
(if you want the second option please go to page 19)

```
private void Start()
{
        if (UseMusclePectoralLeft)
        {
            builtMuscles += pectoralLm + ",";
        }
        if (UseMusclePectoralRight)
        {
            builtMuscles += pectoralRm + ",";
        }
        if (UseMuscleDorsalLeft)
        {
            builtMuscles += dorsalLm + ",";
        }
        if (UseMuscleDorsalRight)
        {
            builtMuscles += dorsalRm + ",";
        }
        if (UseMuscleArmLeft)
        {
            builtMuscles += armLm + ",";
        }
        if (UseMuscleArmRight)
        {
            builtMuscles += armRm + ",";
        }
        if (UseMuscleLumbarLeft)
        {
            builtMuscles += lumbarLm + ",";
        }
        if (UseMuscleLumbarRight)
        {
            builtMuscles += lumbarRm + ",";
        }
        if (UseMuscleAbdominalLeft)
        {
            builtMuscles += abdominalLm + ",";
        }
        if (UseMuscleAbdominalRight)
        {
            builtMuscles += abdominalRm;
        }
        builtMuscles = builtMuscles.TrimEnd(',');
}
```

# VRC OWO System World integration.

```
private void OnTriggerEnter(Collider other)
{
    string currentMuscle = "";

    switch (other.name)
    {
        case pectoralL:
            currentMuscle += pectoralLm;
            muscleTriggered = true;
            break;
        case pectoralR:
            currentMuscle += pectoralRm;
            muscleTriggered = true;
            break;
        case dorsalL:
            currentMuscle += dorsalLm;
            muscleTriggered = true;
            break;
        case dorsalR:
            currentMuscle += dorsalRm;
            muscleTriggered = true;
            break;
        case armL:
            currentMuscle += armLm;
            muscleTriggered = true;
            break;
        case armR:
            currentMuscle += armRm;
            muscleTriggered = true;
            break;
        case lumbarL:
            currentMuscle += lumbarLm;
            muscleTriggered = true;
            break;
        case lumbarR:
            currentMuscle += lumbarRm;
            muscleTriggered = true;
            break;
        case abdominalL:
            currentMuscle += abdominalLm;
            muscleTriggered = true;
            break;
        case abdominalR:
            currentMuscle += abdominalRm;
            muscleTriggered = true;
            break;
        default:
            return;
    }
    if (muscleTriggered)
    {
        if (triggerCount <= 1)
        {
            triggerCount++;
        }
    }
    if (triggerCount == 1)
    {
        triggeredMuscles = currentMuscle;
    }
    else if (triggerCount == 2)
    {
        triggeredMuscles2 = currentMuscle;
    }
    muscleTriggered = false;
}
```

```
// Logic Varibles
private bool muscleTriggered = false;
private int triggerCount = 0;
private string triggeredMuscles = "";
private string triggeredMuscles2 = "";
```

## Multi-Sensation Setting Muscles In order:

This is the code for how we will set the muscles in the order that they were hit. However we need to set some more variables before we can add this. The following explains what variables you need to add.

MuscleTriggered:
This is the logic to advance the count when we actually touch a muscle.

TriggerCount:
This is used to set the muscles in order that they were hit.

TriggeredMuscles:
This is the first that was hit, we set the name of the first muscle that was hit inside of here.

TriggeredMuscles2:
This is the second that was hit, we set the name of the second muscle that was hit inside of here.

(right now this example can only collect two muscles, if you wanted to be able to collect more muscles, you would need to add more variables (10 is the max) and then expand the logic at the bottom of the example for trigger count so we set them into the correct variable.)
This code works by using the “OnTriggerEnter” event,
When this event is triggered we check what was hit, by checking if the name of the object was the same as one of our muscle collider string names. This is done because if there are other triggers in the world that is entered by the player, that will also set off the “OnTriggerEnter” event.
Once this is checked, we add the muscle that was hit to our corresponding “triggeredMuscle” variable. This is done in order of what muscle was hit first

# VRC OWO System World integration.

## Multi-Sensations Using Muscles In Order Building String:

```
private string builtString = "";
private string builtString2 = "";
private string builtString3 = "";
private string multiString = "";
```

```
private string BuildSensationString(string sensation, int frequency, float durationVal, int intensityVal, float rampUp, float rampDown, float exitDelayVal, string muscles)
{
    return sensation + "\","
           + "\"frequency\": " + frequency + ","
           + "\"duration\": " + durationVal + ","
           + "\"intensity\": " + intensityVal + ","
           + "\"rampup\":" + rampUp + ","
           + "\"rampdown\":" + rampDown + ","
           + "\"exitdelay\":" + exitDelayVal + ","
           + "\"Muscles\": {" + muscles;
}
```

```
if (triggerCount == 1)
{
    builtString = BuildSensationString(sensationName, frequency, duration, intensityPercentage, rampUp, rampDown, exitDelay, triggeredMuscles);
    builtString2 = BuildSensationString(sensationName3, frequency3, duration3, intensityPercentage3, rampUp3, rampDown3, exitDelay3, triggeredMuscles);
    multiString = BuildMultiSensationString(builtString, builtString2, builtString3);
    Debug.Log(multiString);
}
if (triggerCount == 2)
{
    builtString = BuildSensationString(sensationName, frequency, duration, intensityPercentage, rampUp, rampDown, exitDelay, triggeredMuscles);
    builtString2 = BuildSensationString(sensationName2, frequency2, duration2, intensityPercentage2, rampUp2, rampDown2, exitDelay2, triggeredMuscles2);
    builtString3 = BuildSensationString(sensationName3, frequency3, duration3, intensityPercentage3, rampUp3, rampDown3, exitDelay3, triggeredMuscles + ","+triggeredMuscles2);
    multiString = BuildMultiSensationString(builtString, builtString2, builtString3);
    Debug.Log(multiString);
}
```

```
private string BuildMultiSensationString(string sensationOne, string sensationTwo, string sensationThree)
{
    if (triggerCount == 2)
    {
        return start + sensationPriority + "," + sensationNameStart + sensationOne + separator + sensationNameStart + sensationTwo + separator + sensationNameStart + sensationThree + end;
    }
    if (triggerCount == 1)
    {
        return start + sensationPriority + "," + sensationNameStart + sensationOne + separator + sensationNameStart + sensationTwo + end;
    }
    return "ERROR";
}
```

### Building the strings:

Before we can get to building our strings we first need to add some more variables. As seen in the image to our left.
( the example can only handle 2 muscles as of right now. If you want more muscles or sensations you need to add more “builtString” variables if you have 6 muscles/sensations you need 6 “builtString” variables. You only need one “multiString” variable.)

This code works by building each sensation in the same manner as a single sensation, then it builds all of the individual built strings into one big multi string. ( you don't need to expand the logic if you aren't expanding the amount of captured muscles or sensations)

For the amount of muscles you want to capture e.g: 6 (10 is the max) you need a “builtString” for each muscle and then an additional one, with this you also need to expand the logic for the “triggerCount”.
if the “triggerCount” = 3 you need 4 “builtString” variables to be set.
If “triggerCount” = 10 you need 11 “builtString” variables to be set.
( you cannot ignore the “triggerCount” logic, as this corresponds to the amount of muscles hit, this can change depending on the game play. It is not set in stone. )

Once the script has made each individual string, it then moves onto building the “multiString” variable.
We need to expand the “triggerCount” logic for this as well for the same reason that we expanded the “triggerCount” logic for the last section of code.
( you cannot ignore the “triggerCount” logic, as this corresponds to the amount of muscles hit, this can change depending on the game play. It is not set in stone. )
Once finished the output format string should look like the images below.
(if it doesn't you have done something wrong. Please go back and check your code and compare it to our documentation)

VRC_OWO_WorldIntegration:[{ "priority":3,"sensation": "SwordDefault","frequency": 60,"duration": 0.3,"intensity": 100,"rampup":0,"rampdown":0,"exitdelay":0,"Muscles": {"abdominal_R":100}},{"sensation": "Sword Bleeding","frequency": 100,"duration": 5,"intensity": 50,"rampup":0,"rampdown":1.4,"exitdelay":0,"Muscles": {"abdominal_R":100}}]

VRC_OWO_WorldIntegration:[{ "priority":3,"sensation": "SwordDefault","frequency": 60,"duration": 0.3,"intensity": 100,"rampup":0,"rampdown":0,"exitdelay":0,"Muscles": {"abdominal_R":100}},{"sensation": "Sword Stab Through","frequency": 50,"duration": 0.3,"intensity": 80,"rampup":0,"rampdown":0,"exitdelay":0,"Muscles": {"lumbar_R":100}},{"sensation": "Sword Bleeding","frequency": 100,"duration": 5,"intensity": 50,"rampup":0,"rampdown":1.4,"exitdelay":0,"Muscles": {"abdominal_R":100,"lumbar_R":100}}]

# VRC OWO System World integration.

## Setting Muscles Predefined:

```
private void Start()
{
        if (UseMusclePectoralLeft)
        {
            builtMuscles += pectoralLm + ",";
        }
        if (UseMusclePectoralRight)
        {
            builtMuscles += pectoralRm + ",";
        }
        if (UseMuscleDorsalLeft)
        {
            builtMuscles += dorsalLm + ",";
        }
        if (UseMuscleDorsalRight)
        {
            builtMuscles += dorsalRm + ",";
        }
        if (UseMuscleArmLeft)
        {
            builtMuscles += armLm + ",";
        }
        if (UseMuscleArmRight)
        {
            builtMuscles += armRm + ",";
        }
        if (UseMuscleLumbarLeft)
        {
            builtMuscles += lumbarLm + ",";
        }
        if (UseMuscleLumbarRight)
        {
            builtMuscles += lumbarRm + ",";
        }
        if (UseMuscleAbdominalLeft)
        {
            builtMuscles += abdominalLm + ",";
        }
        if (UseMuscleAbdominalRight)
        {
            builtMuscles += abdominalRm;
        }
        builtMuscles = builtMuscles.TrimEnd(',');
}
```

```
private string builtMuscles = "";
[SerializeField] private bool UseMusclePectoralLeft;
[SerializeField] private bool UseMusclePectoralRight;
[SerializeField] private bool UseMuscleDorsalLeft;
[SerializeField] private bool UseMuscleDorsalRight;
[SerializeField] private bool UseMuscleArmLeft;
[SerializeField] private bool UseMuscleArmRight;
[SerializeField] private bool UseMuscleLumbarLeft;
[SerializeField] private bool UseMuscleLumbarRight;
[SerializeField] private bool UseMuscleAbdominalLeft;
[SerializeField] private bool UseMuscleAbdominalRight;
```

This is the code for how we will set predefined muscles.
However we need to set some more variables before we can add this. The following explains what variables you need to add.

**BuiltMuscles:**
This holds the muscle string names that we set through the code ( see images on the left)

**Use Muscle"Specific Muscle":**
This is a boolean that we set in the inspector this indicates if you are adding the muscle to the "BuiltMuscles" variable or not.

This code works by using the "Start" event.
When the world loads it will check which muscles were selected using the Use Muscle variables and adds them to the "BuiltMuscles" variable for use in the sensation.

# VRC OWO System World integration.

## Multi-Sensation Using Muscles Predefined Building String:

```
private string builtString = "";
private string builtString2 = "";
private string builtString3 = "";
private string multiString = "";
```

**Building the string:**

As we have already gotten our predefined muscles, all we have to do is build out string. ( keep in mind, we need to expand the amount of “builtString” variables for every sensation we have, e.g: if we have 5 sensations we need 5 “builtstring” variables.

```
builtString = BuildSensationString(sensationName, frequency, duration, intensityPercentage, rampUp, rampDown, exitDelay, builtMuscles);
builtString2 = BuildSensationString(sensationName2, frequency2, duration2, intensityPercentage2, rampUp2, rampDown2, exitDelay2, builtMuscles);
builtString3 = BuildSensationString(sensationName3, frequency3, duration3, intensityPercentage3, rampUp3, rampDown3, exitDelay3, builtMuscles);
multiString = BuildMultiSensationString(builtString, builtString2, builtString3);
Debug.Log(multiString);
```

```
private string BuildSensationString(string sensation, int frequency, float durationVal, int intensityVal, float rampUp, float rampDown, float exitDelayVal, string muscles)
{
    return sensation + "\","
            + "\"frequency\": " + frequency + ","
            + "\"duration\": " + durationVal + ","
            + "\"intensity\": " + intensityVal + ","
            + "\"rampup\":" + rampUp + ","
            + "\"rampdown\":" + rampDown + ","
            + "\"exitdelay\":" + exitDelayVal + ","
            + "\"Muscles\": {" + muscles;
}
```

This code works by calling to our BuildSensationString function, this works the same way the buildSensationString function works in the single sensation. It will call it for every sensation we have. (you'll need to adjust the logic for this code, if you have a different amount of sensations)

```
private string BuildMultiSensationString(string sensationOne, string sensationTwo, string sensationThree)
{
    return start + sensationPriority + "," + sensationNameStart + sensationOne + separator + sensationNameStart + sensationTwo + separator + sensationNameStart + sensationThree + end;
}
```

After we have built all the separate sensation strings, we then need to build our multi-sensation string, this is only done once.

( see images on the left)
Now check to see if the formatting of your output matches the format seen below
(if it doesn't you have done something wrong. Please go back and check your code and compare it to our documentation)

VRC_OWO_WorldIntegration:[{"priority":2,"sensation": "One","frequency": 100,"duration": 0.1,"intensity": 100,"rampup":0,"rampdown":0,"exitdelay":0,"Muscles": {"arm_R":100}},{"sensation": "Two","frequency": 100,"duration": 0.1,"intensity": 100,"rampup":0,"rampdown":0,"exitdelay":0,"Muscles": {"arm_R":100}},{"sensation": "Three","frequency": 100,"duration": 0.1,"intensity": 100,"rampup":0,"rampdown":0,"exitdelay":0,"Muscles": {"arm_R":100}}]


# VRC OWO System World Integration. Credits

***Creators:***

RevoForge
SonoVR

***Document writer / VRC Contributor:***

BassBoostedDuck

***Pre Public Testers:***

kiffin
realmasterlink
nanoade
mrdings
baymaxxxxx
spike_felion
mcsolo
.pepie
rumrobot
wolf_fighter333

kiffin
realmasterlink
nanoade
mrdings
baymaxxxxx
spike_felion
mcsolo
.pepie
rumrobot
wolf_fighter333

***Feel free to contact us on the OWO discord server if you have any issues or bugs.***