Technical Documentation

Information about this Document:
This document is a 'living document', meaning that as time goes on and improvements are made to the various Security Console packages those changes will be reflected in an update to this document. This is to ensure that you are being given accurate reference material as you use our product.

In an effort to make sure that there are a little conflicts as possible all scripts used between the different versions of the packages have different prefixes to distinguish one packages scripts from another. For the sake of clarity, these prefixes have been omitted in this document. Please remember this when searching for the correct script or enumerators.

For example:
If you are using the Four Camera Internal package, and wanted to send the 'ToggleRemoteHUD' command to the 'ManualInput' method you will need to execute the following:

```
FourIntSecConsController.ManualInput(FourIntCameraManualInputCommand.ToggleRemoteHUD)
```

API's

Most of the API's are designed to be used by the integral systems of the "Security system" assets. However some of the API Methods and Delegates are designed so that they can called or subscribed to (respectively) by scripts external to the Security System package sop as to allow further integration into your project and usage design.

This document is designed to help inform as to their proper usage. The API's will be grouped by Objects and Components.

Overall Security System Controller:

StaticFloorMountedConsole:

- SecConsController:
  - **public enum ManualInputCommand:**
    - *Public enumerator*. Used to send the players command to the Security Controller. The list of commands and their descriptions are listed below:
      - RotateLeft:
        - Used to manually rotate the active camera to the left. Will set the active camera into 'manual-mode'.
      - RotateRight:
        - Used to manually rotate the active camera to the right. Will set the active camera into 'manual-mode'.

- RotateUp:
  - Used to manually rotate the active camera to the up. Will set the active camera into ‘manual-mode’.
- RotateDown:
  - Used to manually rotate the active camera to the down. Will set the active camera into ‘manual-mode’.
- ZoomIn:
  - Used to zoom the active camera in. Will only work if the active camera is not already at the highest allowed zoom level.
- ZoomOut:
  - Used to zoom the active camera out. Will only work if the active camera is not already at the lowest allowed zoom level.
- ToggleNV:
  - Used to toggle on/off the Night Vision light on the active camera. This light will only be seen by this camera.
- ToggleLight:
  - Used to toggle on/off the light on the active camera.
- ToggleAutoTrack:
  - Used to toggle on/off the Auto-Track function on the active camera. Note: this will not do anything if the camera has not been setup in the Unity Inspector to use the Auto-Track functionality.
- SwitchCamera1:
  - Used to switch the ‘Active Camera’ control to camera 1.
- SwitchCamera2:
  - Used to switch the ‘Active Camera’ control to camera 2.
- SwitchCamera3:
  - Used to switch the ‘Active Camera’ control to camera 3.
- SwitchCamera4:
  - Used to switch the ‘Active Camera’ control to camera 4.
- SwitchCamera5*:
  - Used to switch the ‘Active Camera’ control to camera 5.
- SwitchCamera6*:
  - Used to switch the ‘Active Camera’ control to camera 6.
- ToggleOnOff:
  - Used to deactivate the keyboard display on the StaticFloorMountedConsole.
- ToggleRemoteHUD:
  - Used to toggle on/off the remote HUD for the player.
- FOVNarrow:
  - Used to set the zoom of the camera to the “Narrow FOV” preset.
- FOVMedium:
  - Used to set the zoom of the camera to the “Medium FOV” preset.
- FOVWide:
  - Used to set the zoom of the camera to the “Wide FOV” preset.

* Only available for the 6-Camera Internal and the 6-Camera External Packages.

- **public bool MustBeFacingConsoleToUse:**
  - *Read Only Property*: This is used by the Security Console Trigger script (see below) to see if the player has to be looking at the static console as well as being within the trigger bounds. This is used in conjunction with the PlayerInteractionAngle (see below). This is set via the Unity Inspector.
- **public bool PlayerInteractionAngle:**
  - *Read Only Property.* This is used by the Security Console Trigger script (see below) to obtain the maximum angle that the player can be looking away from the static console and still be able to access the controls. Player will have to be within the bounds of the trigger collider to be able to use the controls. This is set via the Unity Inspector.
- **public void ManualInput(CameraManualInputCommand command)**
  - *Method*. Used to send a command from the user to the Security System. For a list of commands, please see the ManualInputCommand enumerator above. The commands that can be processed will be restricted depending on if the Remote HUD is open, or the player is standing in front of the Security Console.
- **public void AllowManualInput()**
  - *Method*. This is called by the Security Console Trigger to inform the Security System that the player is now standing in front of the console. This allows the player to be able to interact with the Security System (please see the ManualInput method above).
- **public void DisallowManualInput()**
  - *Method*. This is called by the Security Console Trigger to inform the Security System that the player is no longer standing in front of the console. Calling this method will stop the majority of commands being sent to the ManualInput method being processed.

- StaticConsFeedbackController:
  - **public bool ToggleSystemVisible():**
    - *Method*. This will switch the remote console from on to off, or vice versa. It will return true if there were no errors. If calling this method through your own scripts you will have to call the “ToggleSystemIcon” or “SetSystemIcon” methods manually as well.
  - **public bool SetSystemVisible(bool newVisibleState):**
    - *Method*.This will turn the remote console on if *newVisibleState* is true when called. If it is false then it will turn the remote console off. It will return true if there were no errors. If calling this method through your own scripts you will have to call the “ToggleSystemIcon” or “SetSystemIcon” methods manually as well.
  - **public bool ToggleSystemIcon():**
    - *Method*. This will turn the system icon from on to off, or vice versa. It will return true if there were no errors. This should be called if you manually toggled or set the visible state of the remote HUD.
  - **public bool SetSystemIcon(bool newIconState):**
    - *Method*. This will turn the system icon to on if *newIconState* is true when called. If it is false then it will turn off the icon. It will return true if there were no errors. This should be called if you manually toggled or set the visible state of the emote HUD.

- **public bool SetNVIcon (bool newIconState) / public bool SetLightIcon(bool newIconState) / public bool SetAutoTrackIcon (bool newIconState):**
  - *Method*. All of these will turn the NightVision, Light and AutoTrack symbols on or off depending on the bool parameter passed, respectively.
- **public bool SetCameraActivatedIcon(CameraOption cameraChosen [default = none]**
  - *Method*. This will highlight the chosen icon to show which camera is currently active. Sending CameraOption.None (or leaving the parameters blank and allowing the default values to be used) will turn off all the camera icons.
- **public bool SetFOVSwitchIcon(FOVOption fovOptionChosen [default = none]):**
  - *Method*. This will turn of all FOV option icons on the Remote HUD. Following that, provided that the fovOptionChosen is not the default value of 'none', the icon value passed will turn on.
- **public bool ChangeVideoFeed(int videoFeedNumber, bool newNVState, bool newLightState, bool newAutoTrackState):**
  - *Method*. This is to be called to switch the video feed to a new one. This will handle switching out the render texture material, provided the requested *videoFeedNumber* is within the limits of the render textures assigned to this controller. Please note that you will have to pass in bools to show whether the NightVision, Light or AutoTrack systems are engaged for the correct icons to be displayed appropriately

Camera Controller and Sub-Controllers:

SecurityCameraType[x-x]:

- CameraController:
  - **public bool LightIsOn:**
    - *Read Only Property*. This is used for obtaining the current state of the light attached to this Security Camera.
  - **public bool NVIsOn:**
    - *Read Only Property*. This is used for obtaining the current state of the Night Vision system.
  - **public Transform PivotTransform:**
    - *Read Only Property*. This is used for obtaining a reference to the transform of the GameObject used as the Y pivot (Left-Right rotation) for this camera.
  - **public Transform CameraTransform:**
    - *Read Only Property*. This is used for obtaining a reference to the transform of the GameObject that has the Mesh filter and renderer for the security camera model. This transform acts as the X pivot (Up-Down rotation) of the Security Camera Model itself, and is also the parent of the render camera, light and Night Vision light used by this particular Security Camera.
  - **public Camera CamComp:**
    - *Read only Property*. This is used for obtaining a reference to the render camera used by this Security Camera object.
  - **public float OrigZoom:**

        - *Read Only Property*. This is used to obtain the original zoom level of this render camera. Currently will always return 1f.
    - **public float CurrentZoom:**
        - *Read Only Property*. This is used to obtain the current zoom level of this render camera.
    - **public float OriginalFOV:**
        - *Read Only Property*. This is used to obtain the current Field Of View setting on the render camera. This is used in conjunction with the *CurrentZoom (see above)* to work out the current magnification of the render camera.
    - **public Vector2 CurrentRotation:**
    - *Read And Write Property*. This is used to obtain or update the current recorded rotation of this Security Camera's rotation, which used for calculating angles, speed and rotational vectors. This alone will not update the rotations of any Transforms – other methods should be called to do this. The rotation information is broken up into X and Y rotations. The Y rotation (Left-Right) is according to the PivotTransform (see above), whereas the X rotation (Up-Down) is according to the CameraTransform (see above). If the camera is in it's Auto-Rotate state or it's deactivate state then this will also update the LastRotation Vector2 (see below).
- **public Vector2 LastRotation:**
    - *Read Only Property*. This is used for obtaining the last recorded rotation of this Security Camera whilst in the Auto-Rotate state.
- **public Vector2 DefaultRotation:**
    - *Read Only Property*. This is used for obtaining the original rotation of the Security camera (combined PivotTransform and CameraTransform information) when the scene started. **Please note:** Currently the security camera should have it's PivotTransform and CameraTransform rotations 'reset' to Vector3.Zero before the scene starts otherwise aberrant behaviour will occur. This is a known issue.
- **public bool AutoTrackEnabled:**
    - *Read Only Property*. This will return true if both the Auto-Tracking function and the Auto-Tracking Collider is enabled on this Security Camera.

Please Note:

The following methods listed for Camera Controller are used automatically by the Security Controller. You should not need to adjust them. However, their explanations are as follows:

- **public void InitializeCamera(CameraSettingsContainer newSettings):**
    - *Method*. This method is used to send the settings from the Security Controller to this Security Camera. This happens during the Start() method for the Security Controller. The information sent to this Security Camera includes such information as rotational limits for manual-rotation mode, auto-rotation and auto-track modes; the rotational speeds for auto-rotate/auto-track modes, manual-rotate and 'returning'* modes; minimum and maximum zoom levels; various options available for this Security Camera and the tags of objects that this Security Camera can track.

        ** A 'returning' mode is used when the camera is now returning to a previous rotation that was*

used by the Auto-Rotate mode. Essentially the camera will be "picking up where it left off".

- **public void ZoomIn():**
  - *Method*. This is used by the Security Controller to attempt to 'zoom in'. Since zooming is a manual activity, this will set the camera into 'manual-rotate' mode and reset and 'returning' timers. Zoom limits and zoom speeds are set by the Security Controller when it calls the InitializeCamera method (see above).

- **public void ZoomOut():**
  - *Method*. This is used by the Security Controller to attempt to 'zoom out'. Since zooming is a manual activity, this will set the camera into 'manual-rotate' mode and reset the 'returning' timers. Zoom limits and zoom speeds are set by the Security Controller when it calls the InitializeCamera method (see above).
- **public void RotateLeft() / RotateRight() / RotateUp() / RotateDown():**
  - *Method(s)*. These are called by the Security Controller when the player is manually rotating this Security Camera. By doing so, this Security Camera is set to 'manual-rotate' mode and the 'returning' timers will be set. Rotational Limits and speeds are set by the Security Controller when it calls the InitializeCamera method (see above).
- **public List<GameObject> GetTrackingList():**
  - *Method*. This is used to obtain the list of all targets that can be possibly tracked. The objects were added to this list when they entered the AutoTrack collider, rovided their tag was amongst the list of 'Tags To AutoTrack' that was sent to his ecurity Camera during the InitializeCamera method (see above). Objects ill be moved from this list when they leave the AutoTrack collider.
- **public void SetStateToAutoRotate(bool reverseDir [default = false]):**
  - *Method*. This is used by the Security Controller and the Finite State Machine which handles the modes of the camera to switch from the previous mode (usual 'manual-rotate') to the 'Auto-Rotate' mode. The reverseDir parameter is used to tell this Security Camera to rotate in the opposite direction to that of the direction it was previously 'Auto-Rotating' in.
- **public void SetAutoRotateReversDirection():**
  - *Method*. This is used for manually reversing the direction of the 'Auto-Rotate' mode. Useful for causing a 'broken camera' effect.
- **public void SetStateToReturnWait():**
  - *Method*. This is used to switch the Security Camera into it's 'Return-Wait' mode. This will happen automatically when the camera reaches the end of it's auto-rotation limits when in 'Auto-Rotate' mode. The time before it reversing the direction and continues panning is set during the IntitializeCamera method (see above).
- **public void SetStateToResumeWait():**
  - *Method*. This is used to switch the Security Camera into it's 'Resume-Wait' mode. This will happen when the player has previously applied any 'manual-rotation' to the camera, when the camera is powering back up from being deactivated or when the camera has just left the 'Auto-Track' mode. The delay before starting to resume, and the speed of the rotation to it's last recorded rotation (see LastRotation above) are both set during the InitializeCamera method (see above).
- **public void SetStateToDeactivateState():**

    - *Method*. This is called when the player deactivates the cameras. The time it takes for the Security Camera to return to it's default rotation (see DefaultRotation above) is set during the InitializeCamera method (see above).
- **public void ToggleLight():**
    - *Method*. This is used to simply toggle the light attached to this Security Camera on and off.
- **public void ToggleNightVision():**
    - *Method*. This is used to simply toggle the Night Vision light attached to this Security Camera on and off. **Please note:** this Night Vision light will only be seen by the Camera component attached to this Security Controller.
- **public void ToggleAutoTrack():**
    - *Method*. This is used to toggle auto-tracking on or off. This will only work on Security Cameras that have Auto-Tracking enabled via the Security Controller in the Unity Inspector. These settings are set during the IntializeCamera method (see above) and cannot be changed during runtime.
- **public void SetFOVToWidePreset() / SetFOVToNormalPreset() / SetFOVToNarrowPreset():**
    - *Method(s)*. These methods are used to snap the cameras zoom levels to the presets chosen via the Security Controller in the Unity Inspector. The 'Narrow Preset' should have the highest level of zoom, where the 'Wide Preset' should have the widest angle of view. Although the icons are updated by the Security Controller automatically, when any of the methods are called manually, you will need to make sure that the correct icon is being presented by calling the SetFOVSwitchIcon method on the static and remote HUD feedback controllers, passing the correct parameter of the preset used. For example, if the player has selected the narrow preset then you would call the following:
    RemoteDisplayFeedbackController.SetFOVSwitchIcon(FOVOption.Narrow);

Please Note:

The following methods listed for CameraController are used automatically by the Security Controller or the Finite State Machine. You should not need to access them. However, their explanations are as follows:

- **public void SetRotationAndZoomAsResumeState(Vector2 newRot, float newZoom):**
    - *Method*. This is used by the 'resume-wait' mode to update the rotation and zoom values of the Security Camera. This should not be used except by the 'resume-wait' mode as it will not update the last recorded rotation (see LastRotation above). Furthermore, because this method is updating the rotations directly, the usual speed and rotational limits will not be applied causing a 'snap' to occur.

Camera:

- Camera Night Vision:

- **public bool NightVisionState = true/false;**
  - *Read and Write Property.* This is used to control whether the night vision/Infrared light mode should be on for this camera. Please note that only this camera will detect this light. This script MUST reside on the camera and should not be moved.

Remote HUD System attached to player's camera:

SecurityConsoleHUD02:

- RemoteDisplayFeedbackController:
  - **UnityEvent OnHUDOpen:**
    - *Subscribable Event* . Assign to this Event to call methods when the HUD is opened. This can be done within the Editor, or at runtime.
  - **UnityEvent OnHUDClose:**
    - *Subscribable Event* . Assign to this Event to call methods when the HUD is opened. Again, this can be done within the Editor, or at runtime.

Please Note:

The following methods listed for SecurityConsoleHUD02 are used automatically by the Security Controller. You should not need to access them, however, their explanations are as follows. They are identical to the methods used in the StaticDisplayFeedbackController (see above):

  - **public bool ToggleSystemVisible():**
    - *Method*. This will switch the remote console from on to off, or vice versa. It will return true if there were no errors. If calling this method through your own scripts you will have to call the "ToggleSystemIcon" or "SetSystemIcon" methods manually as well.
  - **public bool SetSystemVisible(bool newVisibleState):**
    - *Method*.This will turn the remote console on if *newVisibleState* is true when called. If it is false then it will turn the remote console off. It will return true if there were no errors. If calling this method through your own scripts you will have to call the "ToggleSystemIcon" or "SetSystemIcon" methods manually as well.
  - **public bool ToggleSystemIcon():**
    - *Method*. This will turn the system icon from on to off, or vice versa. It will return true if there were no errors. This should be called if you manually toggled or set the visible state of the remote HUD.
  - **public bool SetSystemIcon(bool newIconState):**
    - *Method*. This will turn the system icon to on if *newIconState* is true when called. If it is false then it will turn off the icon. It will return true if there were no errors. This should be called if you manually toggled or set the visible state of the emote HUD.

- **public bool SetNVIcon (bool newIconState) / public bool SetLightIcon(bool newIconState) / public bool SetAutoTrackIcon (bool newIconState):**
  - *Method*. All of these will turn the NightVision, Light and AutoTrack symbols on or off depending on the bool parameter passed, respectively.
- **public bool SetCameraActivatedIcon(CameraOption cameraChosen [default = none]**
  - *Method*. This will highlight the chosen icon to show which camera is currently active. Sending CameraOption.None (or leaving the parameters blank and allowing the default values to be used) will turn off all the camera icons.
- **public bool SetFOVSwitchIcon(FOVOption fovOptionChosen [default = none]):**
  - *Method*. This will turn of all FOV option icons on the Remote HUD. Following that, provided that the fovOptionChosen is not the default value of 'none', the icon value passed will turn on.
- **public bool ChangeVideoFeed(int videoFeedNumber, bool newNVState, bool newLightState, bool newAutoTrackState):**
  - *Method*. This is to be called to switch the video feed to a new one. This will handle switching out the render texture material, provided the requested *videoFeedNumber* is within the limits of the render textures assigned to this controller. Please note that you will have to pass in bools to show whether the NightVision, Light or AutoTrack systems are engaged for the correct icons to be displayed appropriately

Examples of use

Using Subscribable events to disable movement whilst HUD is in use:

One use of the subscribable event is to call methods that disable movement when the HUD is opened, and then re-enable movement when the HUD is closed. To do this you will need a script attached to an object in the scene as shown in Fig. 1.

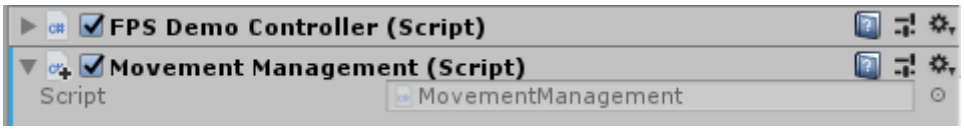

Fig. 1| Movement Management script that will subscribe to the HUD Unity Events

In this script we will have the following code:

```
public class MovementManagement : MonoBehaviour
{
    [HideInInspector] public FourIntFPSDemoController fpsController; //*

    public void OnHUDOpen()
    {
        fpsController.enabled = false;
    }


    public void OnHUDClose()
    {
        fpsController.enabled = true;
    }
}
```

**Please Note: The use of the 'HideInInspector' attribute is added after we have assigned the FPSDemoController reference in the inspector so that we can't accidently remove it.*

From this excerpt we see that we have two methods that can be called: One which will disable our FPSDemoController script (stopping the script from reading inputs and moving the character), and one which will enable the FPSDemoController script (allowing the script to once again to read inputs and move the character).

Back in the editor, we can now assign these methods to our RemoteDisplayFeedbackController so that they can be called whenever the HUD is opened or closed.

We can do this by searching for the SecurityConsoleHUD02 object, selecting it and finding the subscribable events "OnHUDOpen" and "OnHUDClose" as shown in Fig. 2.

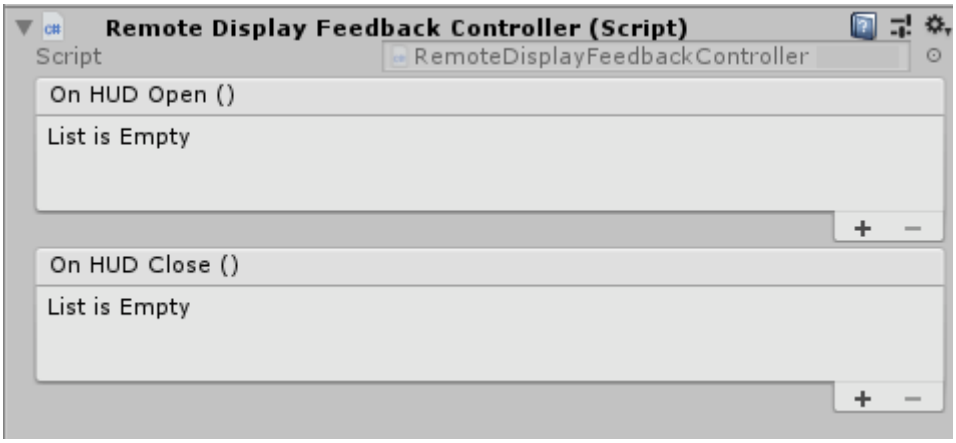

Fig. 2 Unity Events ready to have methods assigned to them.

When we click on the '+' symbol to the bottom right of each Event, we can then drop in an object that has the script that we made ("MovementManagement").

Then we can click on the dropdown box to navigate to the script and method(s) that we want to call, as seen in Fig. 3.

Now whenever the HUD is opened, it will check to see if there are any methods subscribed to it's OnHUDOpen Event, see that the MovementManagement script has it's own OnHUDOpen Method assigned and will call it.

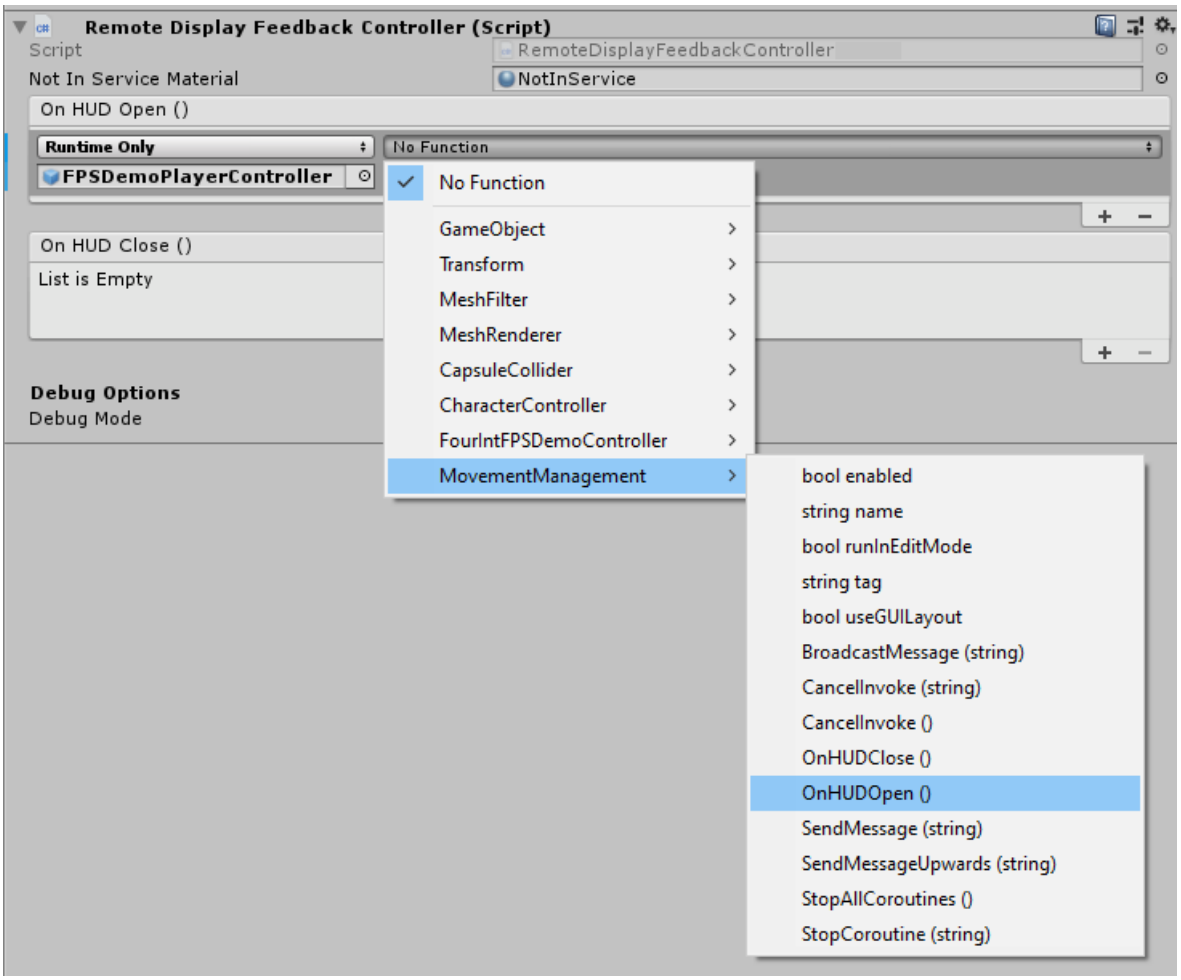

Fig. 3. Selecting the Method to be called by the Event.

You can have many scripts assigned to Unity’s subscribable Events, allowing you to expand gameplay around this core functionality.

Calling the ManualInput method on the SecConsController script:

Although we have supplied the CameraManualInput scripts on the StaticFloorMountedConsole to allow you to simple drop the prefab into the scene to play with, you may find that you want to be able to activate functionality of the Security System without having to always force the player to press the relevant key.

To his end you can call the aforementioned ManualInput method and pass any of the ManualInputCommand enumerator commands to interact with the security console directly.

Hopefully these examples have given you the opportunity to see how you can work with this package that we have put together and integrate it into your current project.

Thank you for supporting us.

We wish you all the best,

Immersive-Games and MadManGames


BROUGHT TO YOU BY.....
IMMERSIVE GAMES .....
&
MADMAN
GAMES