Quick Start Guide

1. Import ICAROS SDK Mobile.unitypackage in Unity.
2. Open Scenes/Examples/Import.scene.
3. It is recommended to save your scene under a different name now. (File -> Save scene as...)
4. [Optional] Import your player model and create a prefab. Link this in the hierarchy at PlayerManager -> PlayerTemplate
5. Import your VR Plugin of choice. Refer to the respective Documentation for help. (For example Vive, Oculus, ...)
6. Link the included CameraRig in the hierarchy at PlayerManager -> VirtualRealityCameraRig
7. Select ICAROS -> Input from the Unity menu bar and choose one of the predefined devices. You may also change their settings or define your own in the Input-settings of Unity. This does not change anything related to the Icaros Controller.
8. [Optional] Put some geometry in your scene as orientation points.
9. Congratulations! You should be able to do your first build!

Next Steps

1. Change the placeholder icon at UISystem -> CompanyLogoTemplate to your own.
2. Adjust some settings at PlayerManager -> LocalPlayer. For Example increase the acceleration of the player. It is not recommended to carelessly change the rotation factors.
3. If you want to build a 'Mirror-App' you can change the state of the application on PlayerManager and UISystem to 'Mirror. Remember to change back to 'Player' after the Build completes. If you are building for PCs change your Architecture to x86_64 in the Unity Build Settings.

Modules and their functions

1. Input
   This module is used to get easily readable input from all input devices supported by the Icaros. The needed setup for use can be seen and copied from the 'Import' scene.

1.1. DeviceManager
   Singleton -> make sure to call functions through DeviceManager.Instance.

   Events:

   - NewDeviceRegistered
     Called for each new device detected. Parameter: the device as IInputDevice.
   - DeviceLost
     Called when the connection to a device is permanently lost. Parameter: the device as IInputDevice.
   - DeviceUsed
     Called when a device gets marked as used. Parameter: the device as IInputDevice.
   - DeviceReconnecting
     Called when a currently used device changes its reconnection state. Parameter: The current State as DeviceManager.reconnectingDeviceState.

- OnDebugMessage
  Called when methods outside the unity framework provide debug information.

Methods:

- UseDevice
  Marks the device as used. Unused devices are inactive in delivering key input. Parameter: the device as IInputDevice.
- GetRegisteredDevices
  Returns a list of all registered IInputDevices.
- GetUsedDevices
  Returns a list of all used IInputDevices.

1.2. IInputDevice

Interface used to get Input without having to know what device is used in reality.
The following Events and Methods are supported:

```
event System.ActionFirstButtonPressed
event System.ActionSecondButtonPressed
event System.ActionThirdButtonPressed
event System.ActionFourthButtonPressed

bool FirstButtonDown()
bool SecondButtonDown()
bool ThirdButtonDown()
bool FourthButtonDown()

event System.ActionFirstButtonReleased
event System.ActionSecondButtonReleased
event System.ActionThirdButtonReleased
event System.ActionFourthButtonReleased

event System.Action<float> xAxisRotated
event System.Action<float> yAxisRotated
event System.Action<float> zAxisRotated
event System.Action<Quaternion> RotationChanged

string GetDeviceTypeID()
string GetDeviceName()
bool IsInUse()
```

2. Player

Optional module that implements a predefined set of controls, collision and a default player model to easily recreate the controls of Icaros Flight. Also contains the basics needed for creating a ‘Monitor-App’.

2.1. PlayerManager

You can change the Spawn of your Player by moving the PlayerManager object in Unity.

All values are directly accessible in the unity editor on the ‘PlayerManager’ object.

You can replace the default vehicle model with your own here.
GameObjec tPlayerTemplate;

Link the respective virtual reality rig you want to use together with your player model here.
GameObject VirtualRealityCameraRig;

Used to determine if this is a player or monitor app.
State state;

This key is used to identify your game on the local area network.
int GameKey;

Used to make different versions of your game incompatible with monitor-apps of different versions.
int GameVersion;
int GameSubVersion;

2.2. LocalPlayer

All values are directly accessible in the unity editor on the ‘Player’ prefab.

Current speed
float MoveSpeed;

Acceleration per button press
float Acceleration;

Multiplies the calculation result for rolls (z-axis rotation)
float RollRotationFactor;

Multiplies the calculation result for pitch (x-axis rotation)
float PitchRotationFactor;

3. Localization

Module used to easily implement different localizations for your project. You can find some predefined values in the file 'Localization/Localization.csv'. Please make sure to always correctly translate those values if you choose to add additional languages.

You can easily update changes made in external programs used for editing the csv file by selecting ICAROS -> Localization -> Load CSV File from the Unity menu bar.

3.1. LocalizationManager

Accessible in the UnityEditor:

LocalizationFileName -> Choose the path/name of your localization file.
DefaultLanguage -> This is the language the game starts at if the user did not choose a preferred language yet.

Easy Access Methods for your Scripts:

void SetLanguage (string languageID)
Change the currently used language to the language of the given ID. It is not recommended to use this to ignore the users settings.

string Get (string tokenID)
returns the text for the given token in the currently selected language.

3.2. LocalizedText

Convenience MonoBehaviour. Can be added to a Unity.UI.Text element via drag and drop and will automatically fill out the text element with the given tokenID using the LocalizationManager.

4. UI

This module is used for easy access to and integration of UI elements in the main menu chain. [This is not meant for or a replacement of in-game UI]

4.1. UISystem

Accessible in the UnityEditor:

UsedCamera -> In case the automatic detection of your VR setup does not work, insert the main camera of your VR-Rig here.
CompanyLogoTemplate -> Insert a Unity-GameObject containing your Companies Logo(s) here. It will be displayed after the Health and Safety Warning Screen.
CompanyLogoDisplayTime -> The time you're the above screen will be displayed for, in seconds.

Easy Access Methods for your Scripts:

void Restart()
Restart thewhole UI chain starting with Health and Safety Warning.

void BackToMainMenu()
Restart the UI chain at the Main menu.

void BackToExternalCameraView()
Restart the UI chain at the external camera view.

void AddLanguageToOptions(string languageID)
Add 'languageID' to the list of available languages. Make sure the chosen ID is already existent in your Localization file. (Example: 'EN' for english)

void RegisterOnPlayFunction(System.Action OnPlay)
Choose a method that listens for presses of the 'play'menu-item by the player.

void RegisterOnMenuItemSelectedFunction(System.Action<string> OnMenuItemSelected)
Choose a method that listens for other menu-item presses. The given string parameter contains the ID chosen for the menu-item when calling RegisterMenuItem and respective variants.

void RegisterMenuItem(string Id, string Title, string Parent)
Register a new menu-item as child element of the ID given in 'Parent'. 'Id' takes the ID of the new element. 'Title' takes the tokenID of the respective text in Localization.

void RegisterPlayMenuItem(string Id, string Title)
Same as the above for registering directly in the play menu.

void RegisterOptionsMenuItem(string Id, string Title)
Same as the above for registering directly in options menu.

void RegisterUnlistedMenuItem(string Id, string Title)
Same as the above for registering menu-items outside the default menu flow. Responsibility to correctly open the menu and direct menu flow lies with the caller of the method.

IMPORTANT(!): All OpenMenu methods may only be called while still in the menu screen. Otherwise refer to BackToMainMenu.

void OpenMenu(string Id)
Open the previously registered menu element with ID ‘Id‘. Make sure this element has children to display.

void OpenMainMenu()
Open the main menu.

void OpenOptionsMenu()
Directly skip to the options menu.

void CloseUI()
Close the UI System. May only be called once the player successfully chose a device (which includes accepting Health and Safety Warnings).

It is not recommended to change or touch any part of the SDK not mentioned in this document.

Multiplayer or other networking modules

To implement Multiplayer with/without Mirroring you can delete the PlayerManager Object from the scene and replace it with your own networking solution. In that case your application is responsible for deciding which instance is the host/client/mirror/player.

The UISystem allows you to change its behavior from Player to Mirror with the ‘state’ dropdown menu. Remember to select this in the scene or set it via script before the UISystem initializes.

In case you do not want to use Unity’s default networking solution you can adjust UISystem -> MonitorController and replace the NetworkIdentity script and related NetworkBehaviour with your networking solution.