# OmniStep 2.0 Developer documentation



# 1 Unity project

The tutorial virtual reality (VR) application “OmniStep” (hereinafter “tutorial”) was created using Unity 2022.3.18f1. Generally, LTS builds offer more stability and longer support than regular builds, making them ideal for projects that require a reliable development environment.

## 1.1 VR template

The Unity project was created using the VR template that comes preconfigured with essential settings targeted specifically for virtual reality. The template incorporates Unity’s XR Interaction Toolkit version 2.5.2, a high-level interaction system designed for creating VR and AR experiences. It offers cross-platform XR controller input via several supported provider plugins, such as Oculus XR or OpenXR. In order to ensure compatibility with a wide range of VR headsets, the OpenXR plugin is used. Unlike the Oculus XR Plugin, which only supports Oculus headsets, OpenXR offers support for the Meta Headsets, VIVE headsets, Valve SteamVR, and more.

### XR Origin

The *XR Origin*, provided in the XR Interaction Toolkit, represents the main VR component in a properly set up scene for virtual reality. It serves as the starting point for configuring the user's interactions with the virtual world, ensuring proper tracking and movement within the virtual space.

### VR Simulator

To be able to walk through the tutorial when a VR headset is not connected, the application switches to a mode that can be controlled using keyboard and mouse. This is enabled via the XR Device Simulator provided by Unity. However, it is intended *strictly for debugging purposes* and should not be considered a substitute for actual VR gameplay, as the user experience may not be optimal or representative of the intended interaction.

## 1.2 KAT Walk integration

In order to make the application work with the KAT Walk treadmills in a seamless way, the KAT Unity Integration SDK version 1.0 was used [1]. It is the official SDK development tool based on Unity XR, developed by KAT VR, and provides compatibility with the following devices: KAT Walk mini S, KAT Loco S, KAT Walk C, KAT Walk C 2. Implementing the support requires the use of the *KATXR Walker* script, which must be added to the GameObject with the *XR Origin* present.

## 1.3 Folder structure

The project mostly follows a standard Unity folder structure, which includes the *Assets* folder as the main directory for all project content. Within it, subfolders are organized by purpose: *Scenes* for scene files, *Scripts* for C# code, *Prefabs* for reusable objects, *Materials* for material assets, *Models* for 3D assets, and *Animations* for animation assets. Additional folders can be found in the project, such as *GeneratedTTS* for files related to the Text-to-Speech package. This structure ensures the project remains clean, organized, and easy to navigate.

## 1.4 Scene hierarchy

The whole tutorial is contained in a single scene called "TutorialScene". As shown in Figure 1, it is organized into several sections for better structure and navigation. The Complete XR Origin Set Up contains all functionality necessary for a VR headset to properly operate.

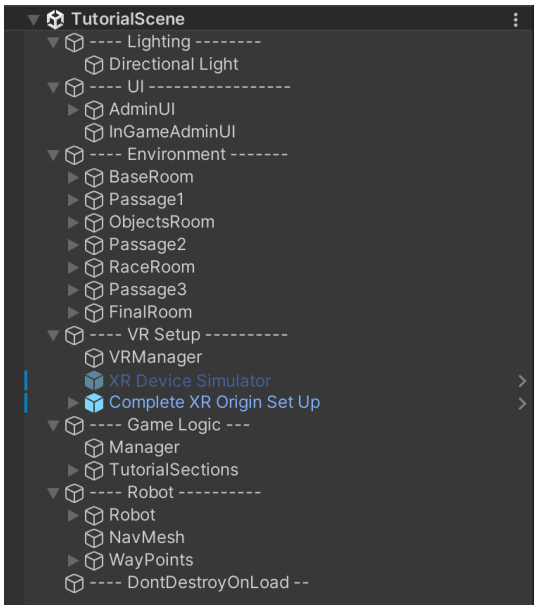


**Figure 1.** Tutorial scene hierarchy.

# 2 Localization package and Text-to-Speech package

The application offers localization in Czech and English. In text form, this is easily implemented using Unity's Localization package [2]. This involves creating a table in Unity, where localized text entries are represented by a string key. For each locale the translation is filled in the table's row belonging to a specific key. This offers an easily extensible solution when more languages are needed.

However, the tutorial provides instructions in both text and verbal form. While the package also supports the use of localized assets such as audio, the process of managing these assets introduces extra work. The main problem is that the audio needs to imported into Unity and manually assigned in the asset localization table. This becomes quite tedious once a larger number of entries is needed. Additionally, any change in the localized text itself means, the corresponding audio clip needs to be regenerated.

In order to improve this process and allow for more dynamic editing, a custom Text-to-Speech package was developed for the tutorial to automate this process. The solution reduces manual workload and allows for quick audio regeneration via a simple button click. It was particularly useful for management of the instructions presented by the robot NPC in the tutorial. The following subsections describe the configuration and implementation of the key scripts.

### Configuration

The custom Unity package can be added to a Unity project via the package manager using a Github URL. The package relies on AI generation and integrates Google Text-to-Speech API [3]. Special Access credentials in the form of a json file are needed and can be obtained from Google Cloud console as described in the above linked documentation. After that, *TTSGeneratorSettings* script must be placed in the Unity scene and filled with the path to the credentials file.

### TextAndAudioManager<T>

The *TextAndAudioManager<T>* generic class serves as a base for the use of the abilities of the package in the project. One such example in this project would be the robot NPC. *RobotTextAndAudioManager* inherits from the base class, supplying itself as the T parameter. This ensures there can be only one *RobotTextAndAudioManager* in the scene as the implementation follows the singleton patter. The script is placed on the robot object in the scene and if correctly setup, as displayed in Figure 2, on button click the audio corresponding to the string localization table will be regenerated.

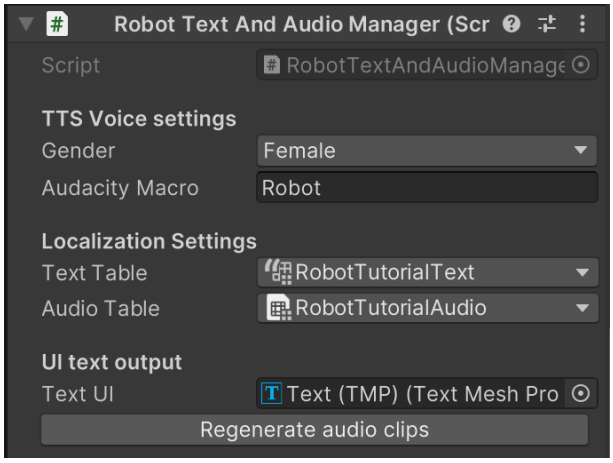


**Figure 2.** "RobotTextAndAudioManager" setup in the tutorial scene.

### TTSGenerator

*TTSGenerator* is a C# script that facilitates the whole automation process. Once the *Regerate audio clips* button is pressed inside the Unity editor, the asynchronous *RegenrateAudio* function is called. It receives the references to appropriate localization tables, path to the json key for Google TTS API, gender of the resulting voice and an optional Audacity macro name if any post-processing is desired.

On the first run, the function exports all of the localized text into json format, as shown in Figure 3. On further use, only entries of edited text are updated marking the field *Regenerate* as true. In this way, only the changes are regenerated and not the whole table.

```
{
  "entries": [
    {
      "key": "good-job",
      "texts": {
        "en": {
          "value": "Good job.",
          "regenerate": true
        },
        "cs": {
          "value": "Dobrá práce.",
          "regenerate": true
        }
      }
    },
    {
      "key": "successful-finish",
      "texts": {
        "en": {
          "value": "You successfully finished the training.",
          "regenerate": true
        },
        "cs": {
          "value": "Úspěšně jste splnili trénink.",
          "regenerate": true
        }
      }
    }
  ]
}
```

**Figure 3.** Structure of the json file for audio generation.

Once the json file is up to date the *google_tts* python script is called, generating all the audio files. The function await completion of the generation task and imports all of the generated audio into the localization asset table under a key, that matches the key of the text entry in the string localization table.

### google_tts

The *google_tts* python script deserializes the previously mentioned json file with the texts. It loops through all the entries and for those marked for regeneration it calls the Google TTS API with appropriate settings such as language and gender. Additionally, if an Audacity macro is specified, it applies it via the programs scripting capabilities utilizing named pipes [4]. For this step to be successful, Audacity must be running on the device and have a macro with corresponding name configured. If that is not the case, the whole generation results in an error that is then printed to the Unity debug console.

## 3 Hand poser

To allow for custom hand grab poses for interactable objects in the tutorial, a hand poser tool was implemented. It is an editor tool that allows for visual hand pose creation. The pose is created in edit mode via joint manipulation and saved to the objects, as shown in Figure 4. When the object is grabbed in the scene during runtime, if a custom set pose is found on the intractable object, it is applied to the hand model. Example of the custom poses in the tutorial are displayed in Figure 5.

The management of this process is handles via a *HandPoserManager* editor script. It provides functionality to create, update, and manipulate poses for both left and right hands.

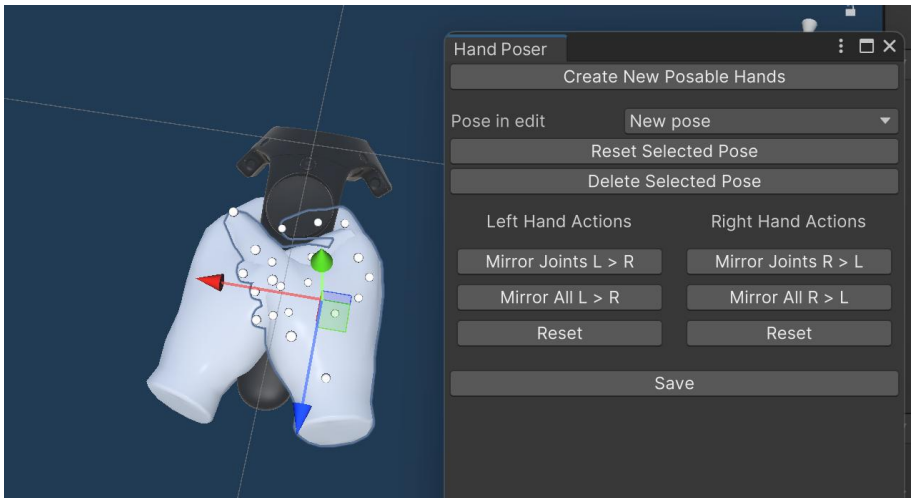


**Figure 4.** Showcase of the hand poser in action.

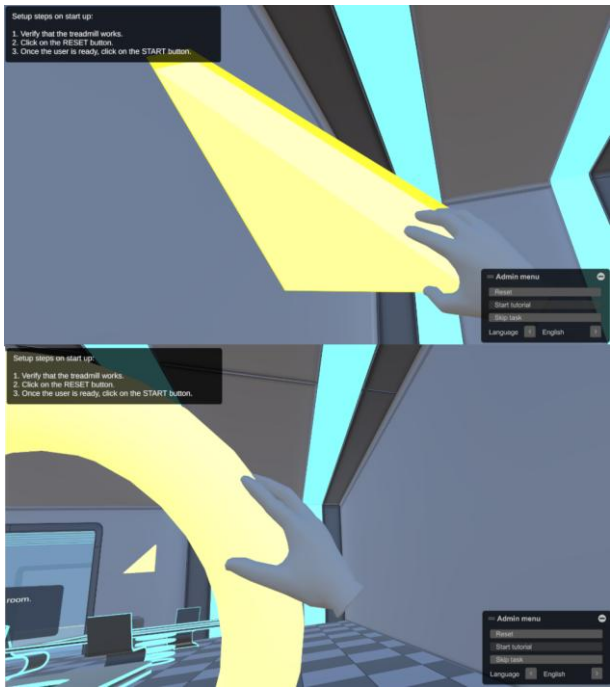


**Figure 5.** Custom hand poses for different objects.

# 4 Tutorial flow implementation

The tutorial flow implementation is based on individual section and task components. However, the division is slightly more granular. This modular implementation architecture ensures flexibility, as sections and tasks can be, to some extent, rearranged, reused, or updated independently, making it easier to iterate on the tutorial content.

The whole tutorial loop operates on an asynchronous basis, passing on cancellation tokens between the components. This enables a skip task functionality based on the use of cancellation tokens. However, this is mainly done for debugging purposes.

## Coroutines versus async/await

In Unity, both **coroutines** and **async/await** are used to handle asynchronous tasks, but serve different purposes and have distinct use cases.

Coroutines are a Unity-specific feature, designed to perform tasks over multiple frames without blocking the main thread. This makes them ideal for managing game-specific tasks such as timed actions, animations, or waiting for events. However, they are not able to return values and do not implement a simple cancellation mechanism like async/await.

On the other hand, async/await is a general-purpose asynchronous programming model in C#. Usually they are used to handle long-running background tasks. However in this case, the main motivation for their use was the simplicity of cancellation, easier implementation of task awaiting inside the tasks and the ability to return values. As Unity does not handle async/await as safely as it handles coroutines, developers must ensure that Unity API calls are executed on the main thread and properly handled via the use of cancellation tokens.

## TutorialSectionsManager class

The *TutorialSectionsManager* script represents the main tutorial loop. It runs a section and awaits its completion before moving to the next. If the application is closed, the cancellation token ensures that the process is properly ended. Figure 6 shows the script's setup within the Unity editor, presenting all the sections of the tutorial. Only one instance of this script must be present in the scene. It is implemented using the singleton pattern, resulting in an error when more than one instance is present. In addition to managing the tutorial flow, the script collects analytics data from each section and saves them to a file at the end of the tutorial. The data contains the time it took for the user to complete each task.

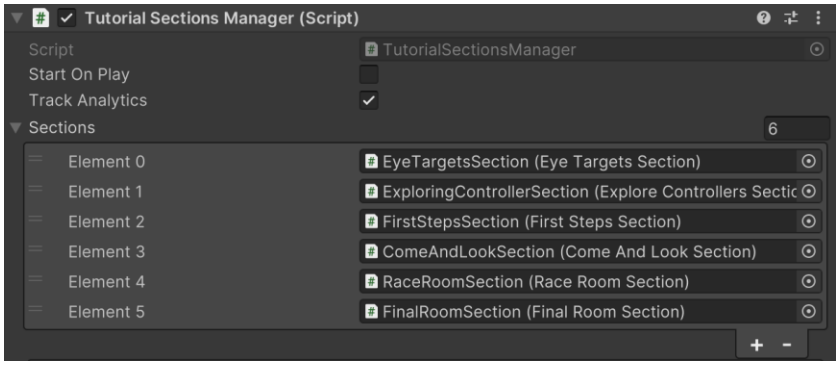


**Figure 6.** "TutorialSectionsManager" setup in the tutorial.

## SectionBase class

*SectionBase* is a common base class for the individual sections within the tutorial. In the current implementation, this component is primarily used for logical organization of tasks and measuring the time of their execution. Additionally, it also propagates proper robot position between tasks, ensuring that on skip, the robot ends up in the desired poistion.

## SectionTaskBase class

The *SectionTaskBase* represents a base class that all of the tasks inherit from. In the implementation, task generally represents a logic encapsulation for one assignment for the user. It provides access to common components, functionality, and handles the cancellation token, ensuring tasks can be interrupted gracefully when the task is skipped or the application closes.

*CleanupOnSkip* is a virtual method that handles the task clean up when it is skipped. In order to ensure proper continuity of the tutorial, if a task is skipped it must apply all the changes as if it was completed. For example if during a task an interactable element appears and stays in the scene, the same must

happen even if the task is skipped before it get to that. It is up the the implementation the the specific task to override this method and apply the necessary changes.

### Extendability

Thanks to the modular architecture of the tutorial, adding new tasks and sections should be relatively easy. New components must derive from the aforementioned base classes and be appropriately added inside the editor to a desired place in a section or the *TutorialSectionsManager* itself. However, it is up to the developer to ensure that proper continuity of the tutorial is preserved. Due to the mostly linear nature of the tutorial, some tasks must precede others for the tutorial to work correctly. For example, the task asking the user to throw a cube cannot come before the cube appears and the user learns how to interact with it. However, it is completely fine to insert a task that encourages the user to try to interact with the cube using their other hand in between.

# 5 Software requirements and installation

When running the application, Windows 10 and Windows 11 are the recommended operating systems. It can be started via the OmniStep.exe present in the Build folder. In order to run the application in VR mode, Steam VR must be installed, and one of the supported VR headsets must be connected. Currently supported headsets include HTC Vive Pro and HTC Vive Pro 2.

The application supports the use of the KAT Walk mini S and KAT Walk C omnidirectional treadmills. Each requires special software to be installed, KAT Industry for KAT Walk mini S and KAT Gateway for KAT Walk C. Having both of these installed on one device seems to cause issues with treadmill input; therefore, it is highly discouraged.

## 5.1 Running application with KAT Walk mini S

When running the application with KAT Walk mini S, it must be done through KAT Industry software, as shown in the document "OmniStep 2.0 How to launch with KAT Industry 2.0.8.pdf" in the Documentation folder.

## 5.2 Running application with KAT Walk C

With KAT Walk C, KAT Gateway software must be running, but the application itself is run separately.

# 6 Assets used

To tie the application's visual design to the theme of modern technology, a futuristic sci-fi aesthetic was chosen. In order to create the desired visuals, free asset packs [5–11] were used for the environment and robot NPC. Several of these assets were modified in Blender to achieve the desired final look presented in the tutorial.

# 7 Acknowledgments

This software documentation is partly based on and includes material originally presented in the master's thesis by Eliška Suchardová, supervised by Michal Sedlák, at Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic [12]. The original text has been edited, updated, and supplemented for the current software documentation. This work was supported by the Center for Virtual Reality Research in Mental Health and Neuroscience (Center for VR Research) at National Institute of Mental Health in the Czech Republic (NIMH CZ).

# 8 References


1. KAT VR. (2023, September 1). *KAT Unity integration SDK*. KAT VR. https://www.kat-vr.com/pages/sdk

2. *Unity Localization package*. (n.d.). https://docs.unity3d.com/Packages/com.unity.localization@1.5/manual/index.html

3. *Google Text-to-Speech API*. (n.d.). https://cloud.google.com/text-to-speech/docs/apis

4. *Audacity scripting*. (n.d.). https://manual.audacityteam.org/man/scripting.html

5. *Asset used 1: Sci-Fi table*. (2024). https://sketchfab.com/3d-models/sci-fi-table-298aa4e4148f4d3b95c6d35efecd52b1

6. *Asset used 2: Loot Box*. (2020). https://sketchfab.com/3d-models/loot-box-24d1d9be93954d3eb7807f8b528d6d98

7. *Asset used 3: Jammo Character*. (2020). https://assetstore.unity.com/packages/3d/characters/jammo-character-mix-and-jam-158456

8. *Asset used 4: 3D Free Modular Kit*. (2024). https://assetstore.unity.com/packages/3d/environments/3d-free-modular-kit-85732

9. *Asset used 5: Star*. (2019). https://sketchfab.com/3d-models/star-5b21e84ddc0342359cd272a9c6dd31cb

10. *Asset used 6: Sci-Fi Target*. (2023). https://sketchfab.com/3d-models/sci-fi-target-cfd36c52c80b4f4ca8d69cec1cf7d293

11. *Asset used 7: success_bell*. (2020). https://freesound.org/people/MLaudio/sounds/511484/

12. Suchardová, E. (2025). *Immersive VR Tutorial Application for Omnidirectional Treadmill: Design, Implementation, and Testing* [Master's thesis, Faculty of Mathematics and Physics, Charles University]. https://dspace.cuni.cz/handle/20.500.11956/197449