ETSI GS ARF 005 V1.1.1 (2022-09)

Augmented Reality Framework (ARF); Open APIs for the Creation and Management of the World Representation

Disclaimer

The present document has been produced and approved by the Augmented Reality Framework (ARF) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGS/ARF-005

Keywords
API, augmented reality, context capturing and analysis, interoperability, real world capture

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Augmented Reality Framework (ARF).

The ISG ARF shares the following understanding for Augmented Reality: Augmented Reality (AR) is the ability to mix in real-time spatially-registered digital content with the real world. The present document specifies the interoperability requirements for Reference Points AR 16 and AR 17 of the reference architecture for AR solutions defined in ETSI GS ARF 003 [1].

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document provides an overview and an introduction to the interface specification for the reference points "AR 16 - World Anchors" and "AR 17 - Reference Objects" of the AR framework architecture [1] developed by the ETSI Industry Specification Group (ISG) for an Augmented Reality Framework (ARF). The actual interface specification is provided as OpenAPI™ specification [3] and forms the baseline for the present document.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI GS ARF 003 (V1.1.1): "Augmented Reality Framework (ARF) AR framework architecture".

[2] ETSI GS ARF 004-2 (V1.1.1): "Augmented Reality Framework (ARF) Interoperability Requirements for AR components, systems and services Part 2: World Storage and AR Authoring functions".

[3] "ARF005 - World Storage API", revision 1.0.0.

NOTE: Available at https://forge.etsi.org/rep/arf/arf005.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] IETF RFC 4122: "A Universally Unique IDentifier (UUID) URN Namespace".

NOTE: Available at https://datatracker.ietf.org/doc/html/rfc4122.

[i.2] IETF RFC 2616: "Hypertext Transfer Protocol HTTP/1.1".

NOTE: Available at https://www.ietf.org/rfc/rfc2616.txt.

[i.3] OpenAPI Specification v3.0.0.

NOTE: Available at https://spec.openapis.org/oas/v3.0.0.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

Augmented Reality (AR): ability to mix in real-time spatially-registered digital content with the real world

AR experience: real time perception of the mixture of the real world and spatially-registered digital content by user senses

AR system: combination of hardware and software that delivers an AR experience

feature: characteristics of a real world element that can be searched, recognized or tracked

NOTE: Features can be of different nature without being limited to visual patterns, UWB, Wi-Fi®, Infra Red or sounds.

pose: combination of position and orientation

reference point: point located at the interface of two non-overlapping functions of the AR framework architecture and representing interrelated interactions between those functions

world anchor: coordinate system related to an element of the real world on which virtual content stays spatially-registered

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
AR	Augmented Reality
ARF	Augmented Reality Framework
CRUD	Create Read Update Delete
GUI	Graphical User Interface
ISG	Industry Specification Group
JSON	JavaScript Object Notation
REST	Representational State Transfer
UI	User Interface
UUID	Universally Unique Identifier
UWB	Ultra Wide Band
WS	World Storage
YAML	YAML Ain't Markup Language

4 Basic structure of the World Storage API for World Anchors and Reference Objects

4.1 Overview

The World Storage API for World Anchors (AR 16) and Reference Objects (AR 17) allow an AR Authoring function to store information about World Anchors and Reference Objects (named in the following document and API as Trackables) in the World Storage at authoring time. This also comprises relative position and orientation information between Trackables and World Anchors. Afterwards, an AR system processes that data at runtime, e.g. through the World Analysis function, to provide an AR experience.

Figure 1 shows a typical architecture used by an AR system. The various authoring processes involve real persons (Author, left-hand side) as well as computing services (right-hand side). While defining a scenario, authors can use the system in different locations, at different times. The API has to assure an easy and elementary access to the World Storage data for creating, managing and deleting World Anchors, Trackables and their connections (World Links).

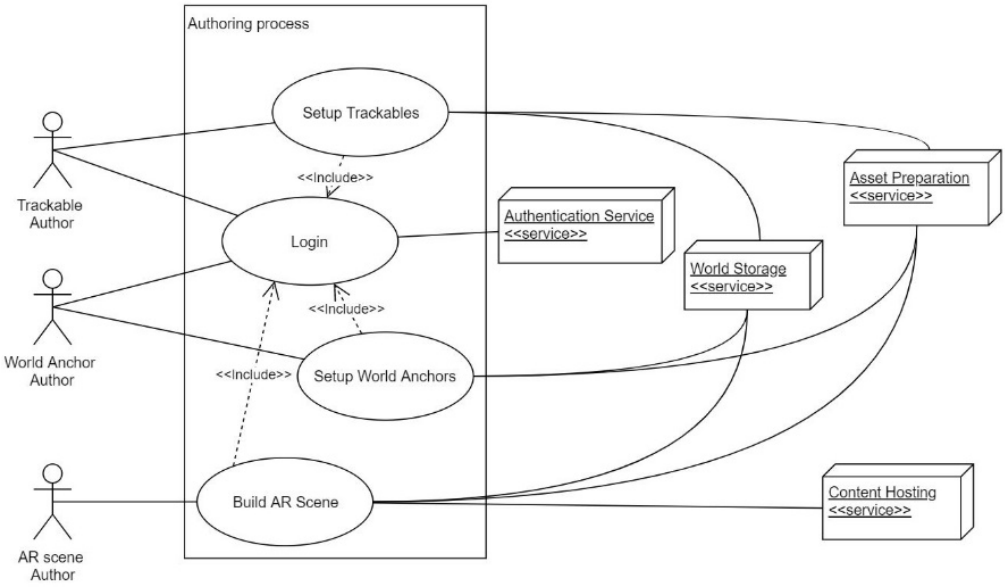


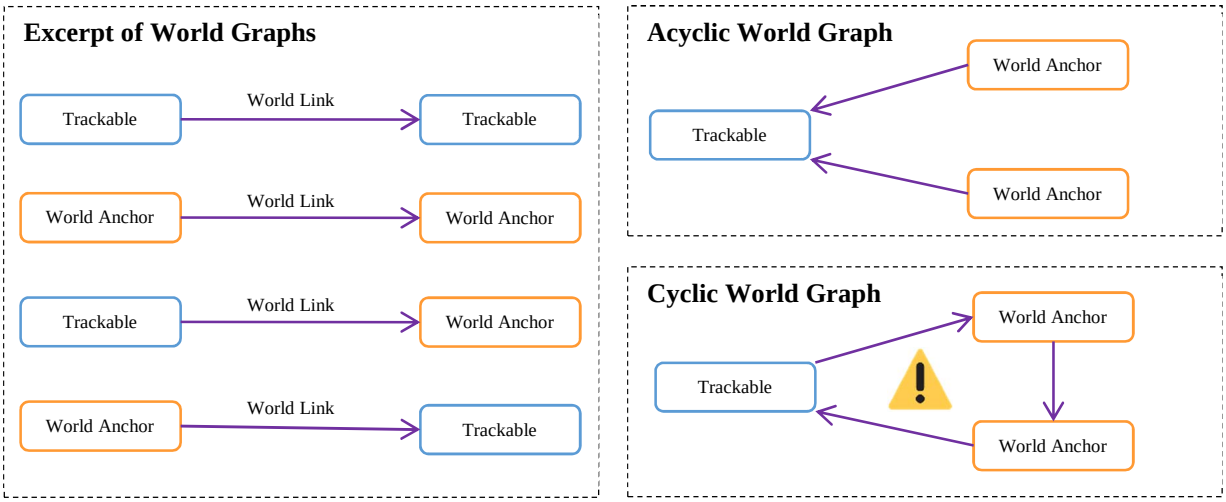
Figure 1: Architecture of a typical AR authoring system

4.2 Elements for world description

A World Graph describes a scene of the real world used at runtime by AR systems to spatially register AR assets with the real world [2]. It consists of Trackables and World Anchors linked together. Trackables shall denote information to track objects of the real world. A World Anchor shall represent a pose in the real world in relation to one or multiple Trackables. World Anchors should be used to attach AR assets at a specific position and orientation in the real world (see Asset Preparation service in Figure 1). AR assets may also be attached to a Trackable directly, e.g. to give the trackable a graphical representation.

To define complex structures and scenarios representing the real world, Trackables and World Anchors should be linked together with World Links. World Links represent the relative position and orientation between Trackables and World Anchors. This allows for retrieving from World Anchors all Trackables and their relative positions useful to estimate the pose of the World Anchors. This World Graph composed of World Anchors, Trackables and their spatial structure may be considered as a graph representing the real world.

The World Storage API may allow relations as shown by Figure 2 (left side).



NOTE: It is recommended to avoid cyclic World Graphs to ease the spatial consistency of Trackables and World Anchors (see Figure 2, right side).

Figure 2: Left: Allowed links between Trackables and World Anchors within a World Graph; Right: Acyclic vs. cyclic World Graph

4.3 Structure of the API

The World Storage API v1.0.0 is available on the ETSI Forge at following URL:

- https://forge.etsi.org/rep/arf/arf005/-/tree/1.0.0

The OpenAPI yaml file is located here [3]:

- https://forge.etsi.org/rep/arf/arf005/-/blob/1.0.0/API/openapi.yaml

The structure of the file is based on the OpenAPI™ 3.0.0 specification [i.3].

The file has 4 main parts:

- Header part (copyrights, info, servers, tags).
- Paths for administration (/**ping**, /**admin**, /**version**).
- Paths for objects (/**trackables**, /**worldAnchors** and /**worldLinks**) - for the description of the actions (REST) and return values.
- Components part - containing the various schemas (in/out parameters) and HTTP responses (errors, warnings).

JSON should be used as coding protocol for parameter and object structures. The functions' return values for single UUIDs shall be non-encoded strings. Other text as well as binary data formats may also be used if some systems request this.

NOTE: Information about the definition and the characteristics of UUIDs can be found in [i.1]. The HTTP semantics are described in [i.2].

The administration paths can be used to test a server, to get the state of a server, and to request the API version.

The API may be tested in the "rendered file" view on ETSI Forge, directly from the repository.

An implementation of the API shall follow the specification in the linked OpenAPI yaml file.

5 Description of the API

5.1 Trackables

The OpenAPI path **/trackables** in the YAML file "openapi.yaml" defines the CRUD operations Create (POST), Read (GET), Update (PUT) and Delete (DELETE) for the Trackables (described in the AR Framework architecture document [1] as the reference point "AR 17-Reference Objects").

Operations:

- Creating and updating a Trackable returns the UUID of the Trackable.
- Accessing one Trackable by UUID returns a JSON document containing the information about that Trackable.
- Accessing all Trackables returns a JSON list of all Trackables.
- Deleting one Trackable (by UUID) removes it from the World Storage.

By deleting a Trackable, the World Storage server shall update all World Links referring this Trackable.

A server shall implement all operations for managing Trackables. Anyway, it may refrain from accepting operation calls for creating updating and deleting Trackables to prevent unauthorized changes in the database.

NOTE: See https://forge.etsi.org/rep/arf/arf005/-/blob/1.0.0/API/openapi.yaml#L76.

5.2 World Anchors

The OpenAPI path **/worldAnchors** in the YAML file "openapi.yaml" defines the CRUD operations Create (POST), Read (GET), Update (PUT) and Delete (DELETE) for the World Anchors (described in the AR Framework architecture document [1] as the reference point "AR 16-World Anchors").

Operations:

- Creating and updating a World Anchor returns the UUID of the World Anchor.
- Accessing one World Anchor by UUID returns a JSON document containing the information about that World Anchor.
- Accessing all World Anchors returns a JSON list of all World Anchors.
- Deleting one World Anchor (by UUID) removes it from the World Storage.

By deleting a World Anchor, the World Storage server shall update all World Links referring this World Anchor.

A server shall implement all operations for managing World Anchors. Anyway, it may refrain from accepting operation calls for creating updating and deleting World Anchors to prevent unauthorized changes in the database.

NOTE: See https://forge.etsi.org/rep/arf/arf005/-/blob/1.0.0/API/openapi.yaml#L219.

5.3 World Links

The OpenAPI path **/worldLinks** in the YAML file "openapi.yaml" defines the CRUD operations Create (POST), Read (GET), Update (PUT) and Delete (DELETE) for the World Links.

Operations:

- Creating and updating a World Link returns the UUID of the World Link.
- Accessing one World Link by UUID returns a JSON document containing the information about that World Link.

- Accessing all World Links returns a JSON list of all World Links.
- Deleting one World Link (by UUID) removes it from the World Storage.

A server shall implement all operations for managing World Anchors. Anyway, it may refrain from accepting operation calls for creating updating and deleting World Anchors to prevent unauthorized changes in the database.

NOTE: See https://forge.etsi.org/rep/arf/arf005/-/blob/1.0.0/API/openapi.yaml#L362.

Annex A (informative): Example Usage of the World Storage API

A.1 Introduction

This annex provides an example (Validation Application 'Museum') illustrating the application of the World Storage API file specified in the present document. The example is kept simple and uses a frequent scenario where a physical object (in the real world) is enriched with some virtual content (placed in the AR scene).

A.2 Specification of the Validation Application 'Museum'

An ordinary object (e.g. a statue in a museum) is to be enriched (documented) with some virtual content via the help of AR techniques.

Use case:

A staff member (generally speaking an author) of the museum decides to augment a statue with a mesh (a geometric model) and a text label (a GUI element) and will propose an access to this virtual content to his visitors from two positions, from the left and the right side of the statue.

For this use case, the author decides to use the World Storage API to create a World Graph of the scenario according to the specification at hand. The use case representing an application of the World Storage API (Validation Application) is visualized in Figure A.1. For this, the author needs to create a structure using Trackable, World Anchor and World Link objects. The author should delegate the various tasks to different persons like a Trackable Author, a World Anchor Author or an AR Scene Author.

Technical specification:

a) Two AR markers are needed (Trackables).

b) Two locations are needed to position virtual content (World Anchors).

c) Some spatial relations between the markers and the locations are needed (World Links).

Task descriptions:

1) The author places AR markers on the left and right side of the statue and registers them as Trackables.

2) For the first content (mesh of the statue) the author decides to place a first World Anchor on the top of the statue's pedestal knowing that the geometry model has its origin here.

3) The second content will be a description (GUI) for the nose of the statue, so the author places the second World Anchor on the nose.

4) Because the content will be accessed by both AR markers, the author first connects the pedestal World Anchor to the two Trackables.

5) Then, the author connects the nose World Anchor to the pedestal World Anchor.

6) Finally, the mesh of the statue is attached to the pedestal World Anchor, and the GUI element is attached to the nose World Anchor (this information is not saved in the World Storage but is defined in the AR scene of the application).

A.3 World Graph of the Validation Application

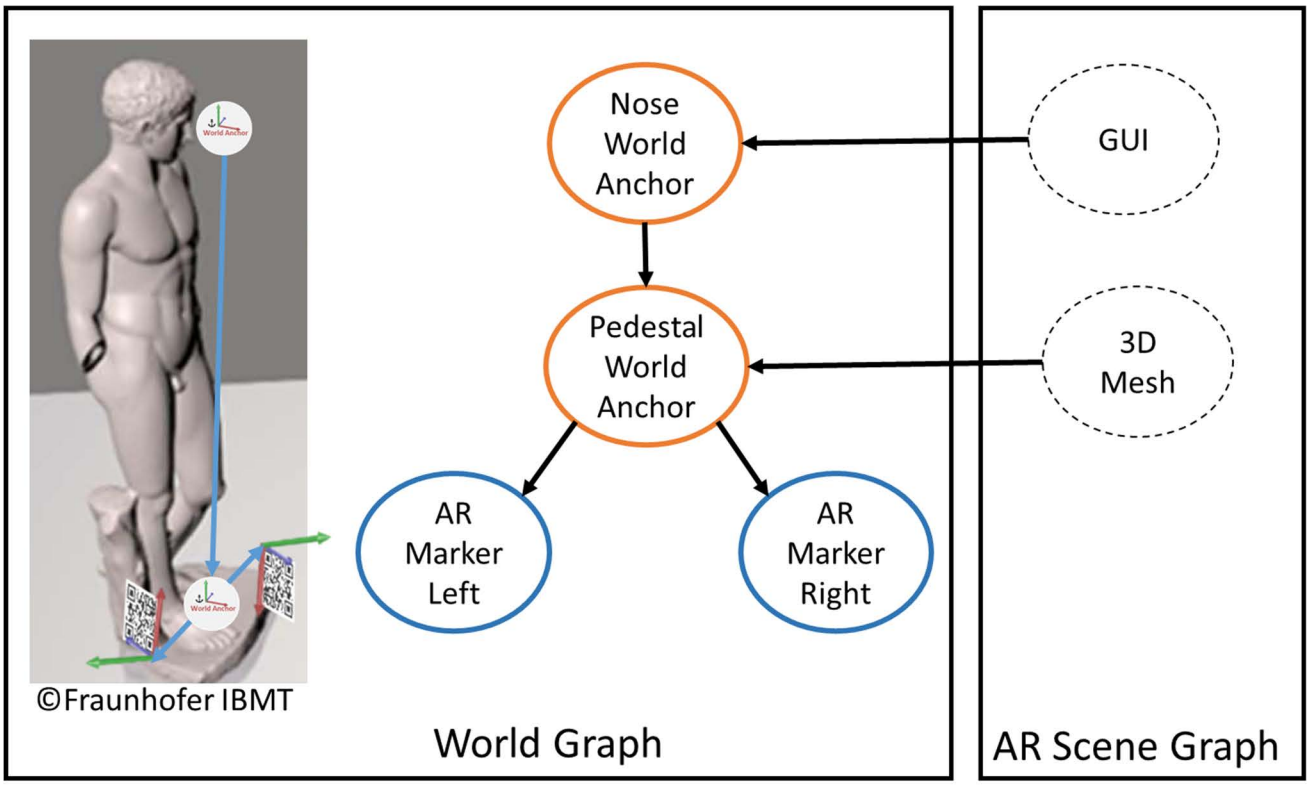


Figure A.1: Simple World Graph for the Museum use case

Figure A.1 represents the Museum use case:

- The left picture shows the locations of the different World Elements.
- The World Graph (middle of the picture) shows the objects and their relations. The arrows represent the WorldLink. Trackables are drawn in a blue circle and World Anchors in orange ones.
- The right part in the figure (AR Scene Graph) represents the AR content that the user will see in the final application.

NOTE: The definition and integration of AR content is not part of the present document.

A.4 Example Usage of the API (Validation Application)

This clause contains examples of REST requests that can be sent by the authoring tool and received by the World Storage server with its responses. They match the request exchanged in the context of the use case described in clause A.2. The author (the Trackable Author and World Anchor Author creating the scenario, see Figure 1) is using the World Storage API or an authoring tool implementing it (e.g. Unity), to instantiate the World Graph as shown in Figure A.1 in which he will set up elements of the real world corresponding to those of its museum.

1) "The author places AR markers on the left and right side of the statue and registers them as Trackables".

Once the author sets up a Trackable in the World Graph, a REST request is sent to save that trackable in the World Storage. Below is the corresponding REST request sent through Postman, an online tool to test APIs:

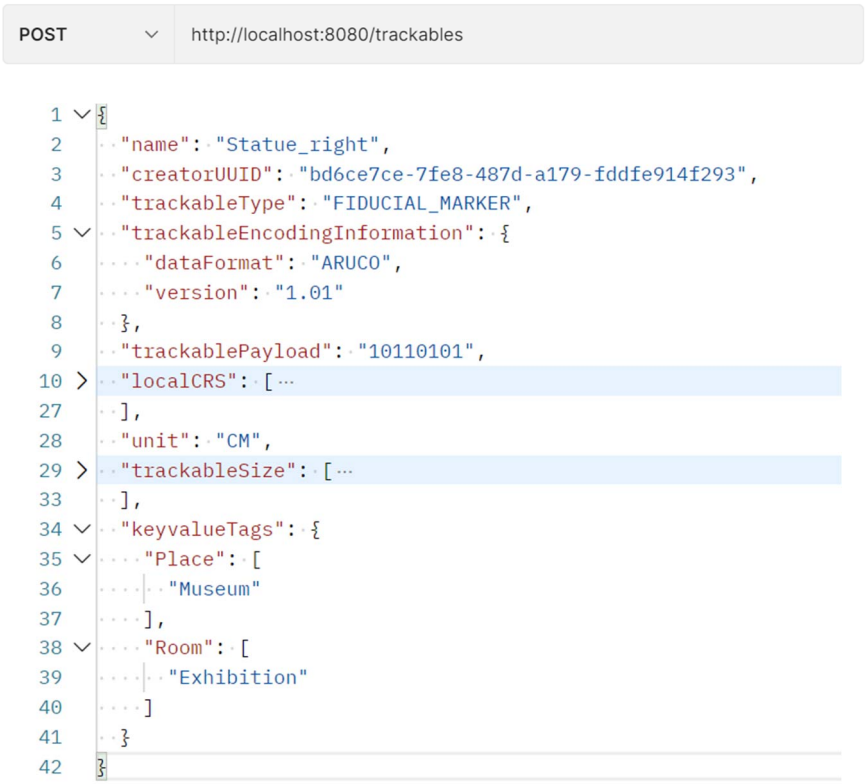


Figure A.2: POST request sent to a local hosted instance of the World Storage

The top part of Figure A.2 is the URL on which the request is sent, and the part below is the body of the request, a JSON containing all the Trackable's attributes. In case of success on the server's side, the response is the UUID of the newly added trackable, otherwise an error is sent by the server. Here is the response from the server in Figure A.3:

```
"6afbc64a-c523-47b7-b472-4a296aeba77c"
```

Figure A.3: POST request response: the ID of the newly added Trackable

1) "For the first content (mesh of the statue) the author decides to place a first World Anchor on the top of the statues pedestal knowing that the geometry model has its origin here."

2) "The second content will be a description (GUI) for the nose of the statue, so the author places the second World Anchor on the nose."

Once the author has set up his two anchors (task 2 and 3) in an authoring tool, two POST requests are sent to save each of them in the World Storage similar to task 1. The URL for these requests would be **http://{servername}:{port}/worldAnchors** and the bodies of the requests contain all the information about the two World Anchors in JSON format. The response of each request is the ID of the anchors in the World Storage, otherwise an error is sent by the server:

1) "Because the content will be accessed by both AR markers, the author first connects the pedestal World Anchor to the two Trackables."

2) "Then, the author connects the nose World Anchor to the pedestal World Anchor."

To create those links (task 4 and 5) between the elements of task 1, 2 and 3, REST requests are sent to create World Links. For this, POST requests are sent to **http://{servername}:{port}/worldLinks**. The bodies of the requests are World Links, they hold the IDs of the two elements they need to connect in the world graph respectively. They also contain other information needed by the author. The ID of the World Link is received if it was added to the World Storage successfully, otherwise an error is sent by the server.

1) "Finally, the mesh of the statue is attached to the pedestal World Anchor, and the GUI element is attached to the nose World Anchor (this info is not saved in the World Storage but is defined in the AR scene of the application)."

For this last task, the AR scene author (see Figure 1) has to request the server to retrieve elements produced in tasks 1 to 5. The author has both World Anchor IDs available in the authoring tool. Two GET requests are sent to retrieve the information of those World Anchors. Since those are element-specific requests, the author sends as a parameter the UUID of the two World Anchors on the URL **http://{servername}:{port}/worldAnchors/{WorldAnchorID}**, the parameter is directly included in the URL.

As an example, if the author wants to retrieve all the information about the World Anchor with the id **501f76e2-ac8a-0594-6d07-ea4d029e2053** on the locally hosted World Storage (port 8080), the author sends a GET request on the URL **http://localhost:8080/worldAnchors/501f76e2-ac8a-0594-6d07-ea4d029e2053**.

There are also other endpoints available for the author to manage the World Graph. There are also endpoints to modify and delete elements. For instance, the user can send a GET request to the URL **http://{servername}:{port}/worldAnchors** to get all the World Anchors currently stored in the World Storage in a JSON format, this is also possible for Trackables and World Links on their respective URLs. This can be used to display the World Graph as a whole in any authoring tool implementing this World Storage API.

Annex B (informative): Demonstration of the Authoring of the Validation Application 'Museum' with Unity Plugins

B.1 Introduction

This annex provides a demonstration of the authoring of the Validation Application. It illustrates the integration of the World Storage API in the Unity Editor and uses the Unity plugins created from the auto generated C# code of the OpenAPI generator. The implementation of this Editor systems (user interfaces and functions) was kept elementary and mostly technical. It uses the Validation Application scenario 'Museum' described in Annex A, where a physical object (in the real world) has to be enriched with some virtual content (placed in the AR scene).

The project can be loaded from the ETSI Git-Labs.

See following URL: https://labs.etsi.org/rep/arf/world-storage-api-helpers/unity-world-storage-editor.

The Unity plugins can be found here: https://labs.etsi.org/rep/arf/world-storage-api-helpers/unity-world-storage-package.

NOTES:

1) The solution proposed here does not demonstrate all possible ways for handling World Storage elements and visualising things in the scene hierarchy and graph view of the Unity Editor.

2) This annex will not be attended to be a full documentation. It is a base information and gives the user an idea how to use the demonstrations.

3) The ordering of the steps below may be different, depending on the scenario.

B.2 Authoring the World Storage with Unity Editor windows

This first implementation demonstrates the communication and management of World Storage data with the common Unity windows UI system (as used by most of the Unity inspector modules).

Step 1: Defining a user

First, the implementer defines a user (see Figure B.1). For a simple system a single user will be fine, but more complex projects need more. Managing the role of the users can also be helpful. In Figure 1, three roles are mentioned:

i. the Trackable Author;
ii. the World Anchor Author; and
iii. the AR Scene Author.

A correct management of users and roles is important because entries in World Storage servers are bound to individual persons. To define a user, the implementer should open the inspector windows based on the corresponding Unity Scriptable Object.

Here, the demo username is ***ARF User*** and his creator UUID is '5090fd9f-64bf-4e06-843f-26aaf2eb8e40'. Only the ***UUID*** is used in the API. In this version of the API, to enter a role is not mandatory (user management is out of the scope of the present document).

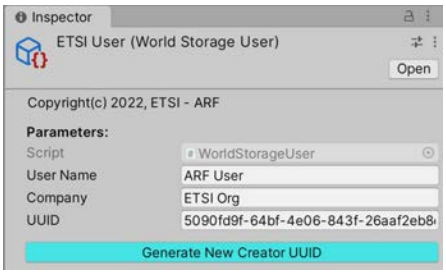


Figure B.1: The setting window for the user ID (API parameter is creatorID)

Step 2: Setting up the World Storage server

Once the user was created, the implementer setups a World Storage Server (Figure B.2). The fields ***Base Path*** and ***Port*** are used to build the address for HTTP requests. If needed, the server should be accessible from outside a company. The user Scriptable Object can be entered in the last field ***Current User*** .Information about the HTTP protocol can be found in IETF RFC 2616 [i.2].

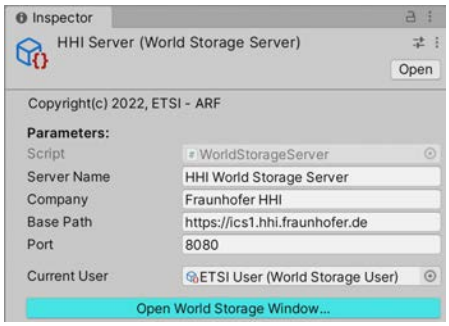


Figure B.2: The setting window for the server

Step 3: Testing and getting the content of the Worlddatabase

Figure B.3 shows the main ARF window after the implementer had (i) tested the server accessibilities (ping, state), (ii) requested the version of the World Storage API (v1.0.0) and (iii) requested the World Elements objects. Note that the group filter '*Museum'* is activated. So, only elements with a corresponding key-value tag named '*Group*' are shown.

By clicking the button ***Create New*** (for elements area) or ***Edit…*** for a single element, the implementer will access to the three different parameter windows (see the next sections). Removing an element is secured while a message window will appear.

NOTE: In all window forms, Trackable elements use a *red* colour, World Anchors an *orange* colour and World Links a *violet* colour. This is also true for the Graph View module (see clause B.3).

Figure B.3: The main Editor window

Step 4a: Editing the parameters for Trackables (red accent)

The form in Figure B.4 allows an implementer (e.g. a Trackable Author) to edit and change parameters for Trackables.

After having changed some values, the implementer clicks on the button ***Save*** to send (push REST action) the modification back to the server. If needed, he can also update the visual (see later in this subchapter) by clicking onto the blue button ***Generate/Update GameObject***.

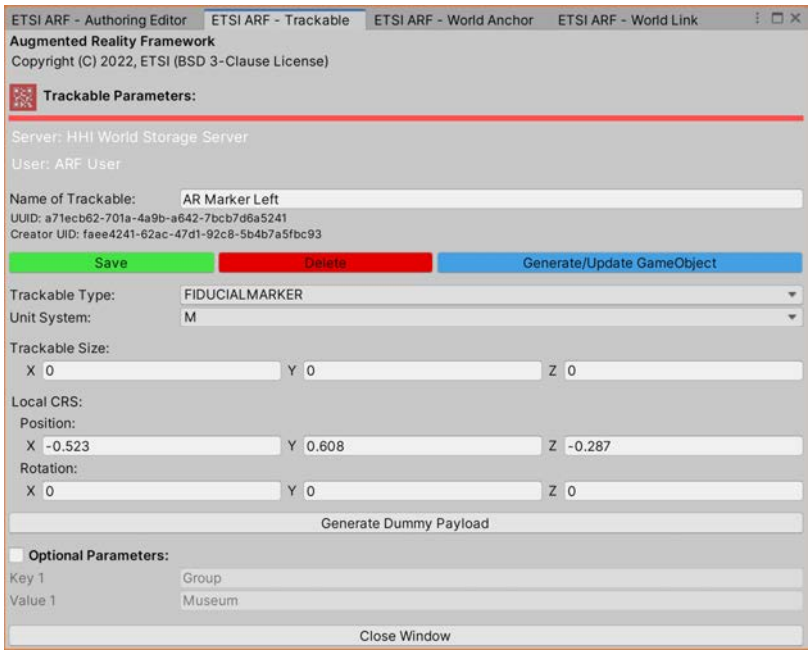


Figure B.4: The editor window for trackables

Step 4b: Editing the parameters for World Anchors (orange accent)

This form in Figure B.5 allows an implementer (e.g. a World Anchor Author) to edit and change parameters for World Anchors.

This step is equivalent to step 4a: After having changed some values, the implementer clicks on the button ***Save*** to send (push) the modification back to the server. If needed, he can also update the visual (see later) by clicking onto the blue button ***Generate/Update GameObject***.

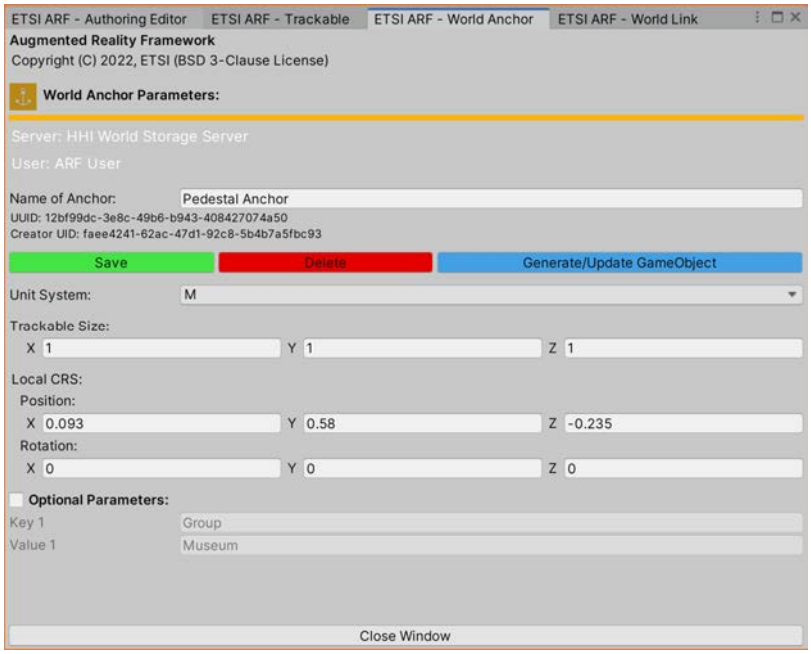


Figure B.5: The editor window for anchors

Step 4c: Editing the parameters for World Links (violet accent)

This form in Figure B.6 allows an implementer (e.g. a World Link Author) to edit and change parameters for World Links.

To make a link between two elements (***Parent/Child*** relations, API internally the parameters are *from/to*) the implementer enters the ID of both of them in the ***UUID*** fields and sends a request to the server (button ***Request***). Then, the forms will be actualized and the element's names are showed in the filed ***Name***.

Alternative linkage: To facilitate the setup of links between elements, the implementer can also select in the scene hierarchy of the Unity Editor two *gizmos* elements (*ctrl-key* has to be used while selecting) and then click the button ***Use From-To Objects from Scene Selection***. The order of the selection is taken to assign them to the parent/child properties. For this, some visual objects (gizmos) of World Elements need to be created to view Trackables, World Anchors and World Link in the current Unity scene (step 5 explains this).

This step is equivalent to step 4a: After having changed some values, the implementer clicks on the button ***Save*** to send (push) the modification back to the server. If needed, he can also update the visual (see later) by clicking onto the blue button ***Generate/Update GameObject***.

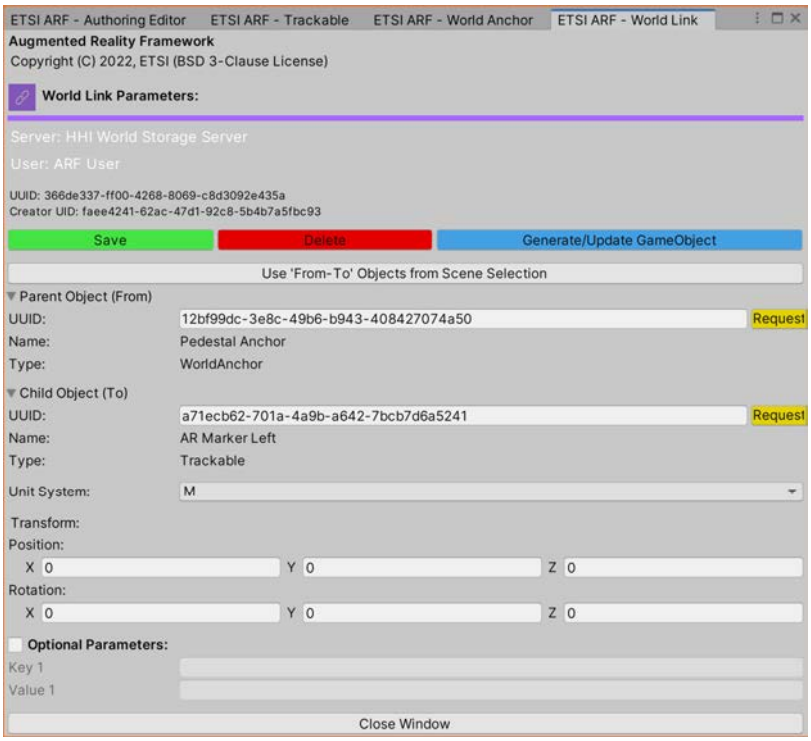


Figure B.6: The editor window for links

Step 5: Generating the Unity scene *gizmos* for the elements

With a click on the button ***Generate/Update GameObject*** (note: the button exists in all editing windows, see steps 4a/b/c) the implementer (e.g. a AR Scene Author) can generate a visualization of the World Elements (*gizmo*), direct in the Unity scene window. Figure B.7 shows all the elements needed for the Validation Application, the '*Museum*' use case.

NOTE: The room and the statue are 3D mesh assets and represent the physical environment and exhibit (digital twin of the real world). They are visualised to give a better understanding of the World Graph. These objects as well as all the gizmos are not rendered in the final AR application.

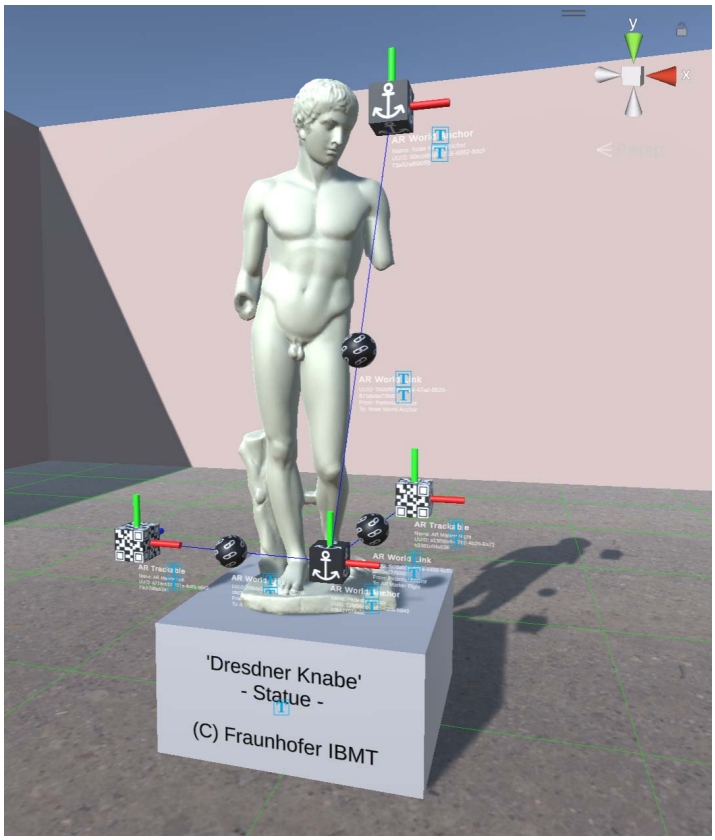


Figure B.7: The 'Museum' use case with the statue and the generated Elements gizmos

The GameObjects are a copy of their corresponding Unity prefabs. The prefabs are registered in the Scriptable-Object named '*ARFPrefabs*' (a resource asset). The gizmos are placed in the scene under '*ARF Visuals*'. Saving World Elements does not modify their gizmo. The implementer has to refresh the gizmo by re-clicking on the button ***Generate/Update GameObject***.

Figure B.8 shows the setup windows for the Elements visuals. Other prefabs for the three World Elements may be defined depending on the scenario.

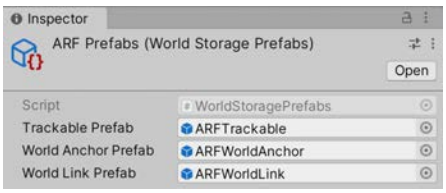


Figure B.8: The window for the visual prefabs (for the gizmos)

B.3 Authoring the World Storage from the Graph View window

This second implementation demonstrates the communication and management of World Storage data in a graphical style, the graph view, as modern Unity modules use to describe relations between objects.

Opening of the Graph View window

In order to create the graph corresponding to the museum use case, the implementer has the possibility to open the graph view from the unity editor toolbar. In the ***ARFWorldStorage*** tab, the implementer can select the ***Edit Graph…*** option, see Figure B.9.

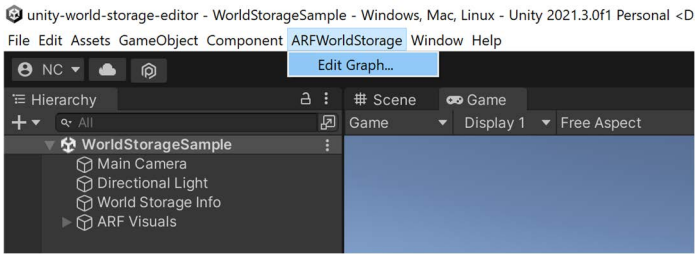


Figure B.9: Opening the World Graph via the Unity editor toolbar

Selection of a World Storage

Once the World Graph window is opened, the implementer selects the desired World Storage to display and then has access to all the authoring functions of the World Storage in a graphic way. By right-clicking on the window, the implementer can create a Trackable and/or a World Anchor, can save the local changes to the online World Storage and can reload the graph from the World Storage (discarding the local changes).

In Figure B.10, the implementer opened a local hosted World Storage server opened on port 8080. The World Storage has no elements yet and the implementer right clicked, getting access to the authoring options.

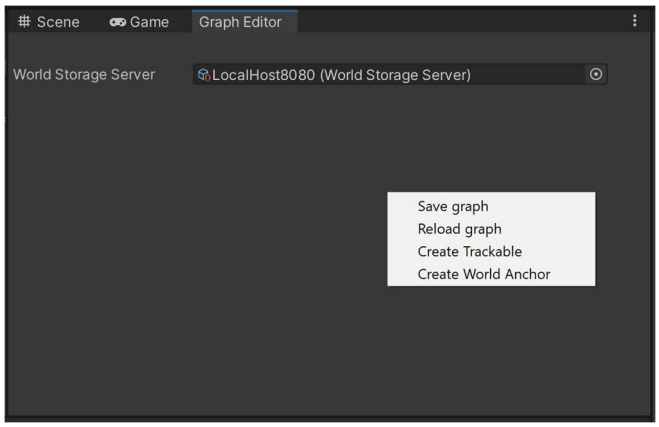


Figure B.10: The World Graph window

Creating a World Storage graph

The implementer may now create the graph corresponding to the '*Museum*' use case. When the implementer creates a new element (World Anchor or Trackable) it is created with default values. It is up to the implementer to modify the element's attributes as wished, by using the element's editor. In Figure B.11, the implementer created a Trackable and double clicked on the ***Graph Node***, opening this element's editor (right side).

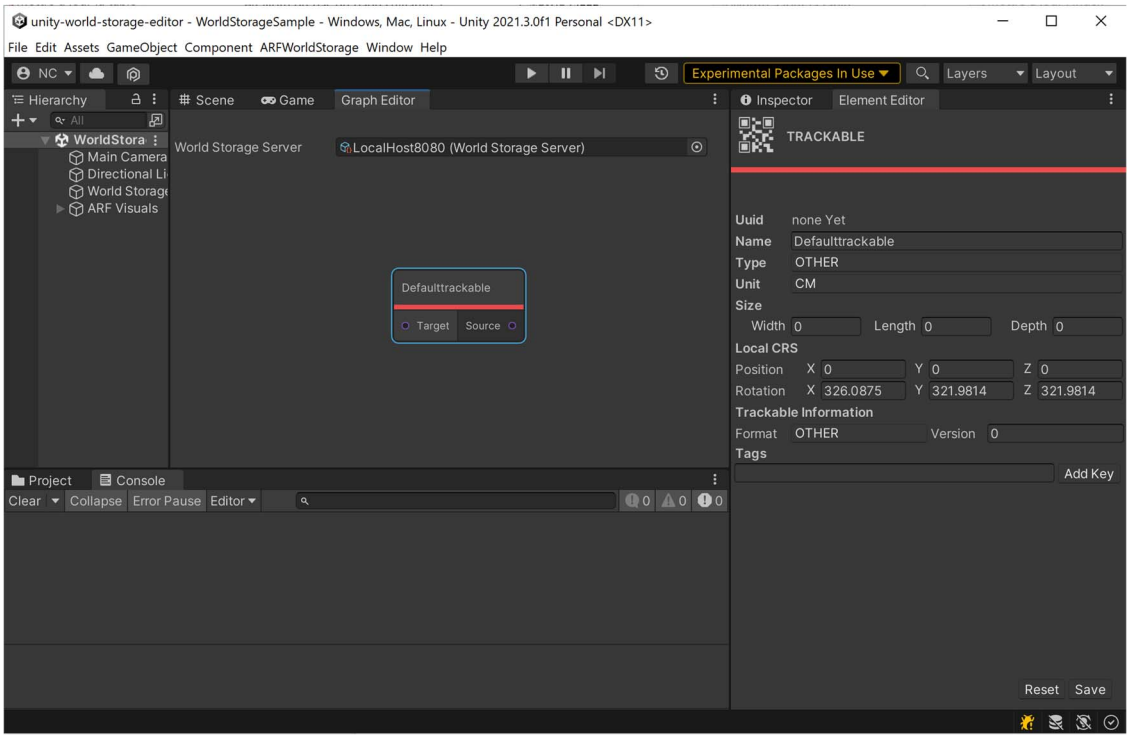


Figure B.11: A newly created Trackable and its default values

Step 1: The first step for the implementer (e.g. a Trackable Author) is to create the two Trackables attached to the sides of the statue. For this, the ***Create Trackable*** option is selected from the contextual menu (right click menu) twice to create the two Trackables, then double click on each of them to modify their attributes according to what he needs to have.

Step 2: The second step is to create the two World Anchors, so the implementer (e.g. a World Anchor Author) selects ***Create World Anchor*** this time to create the two World Anchor nodes and, once again, double clicks on them to modify their attributes.

Step 3: Once it is done, the implementer (e.g. a World Link Author) wants to link them. To do so, he or she can drag and drop from an element's port to another's (matching a Source (from) port to a Target (to) port). To define the 3D Transform that the link represents, the implementer double clicks on the link and then modify the attributes in the editor window again. The graph is now complete (see Figure B.12)

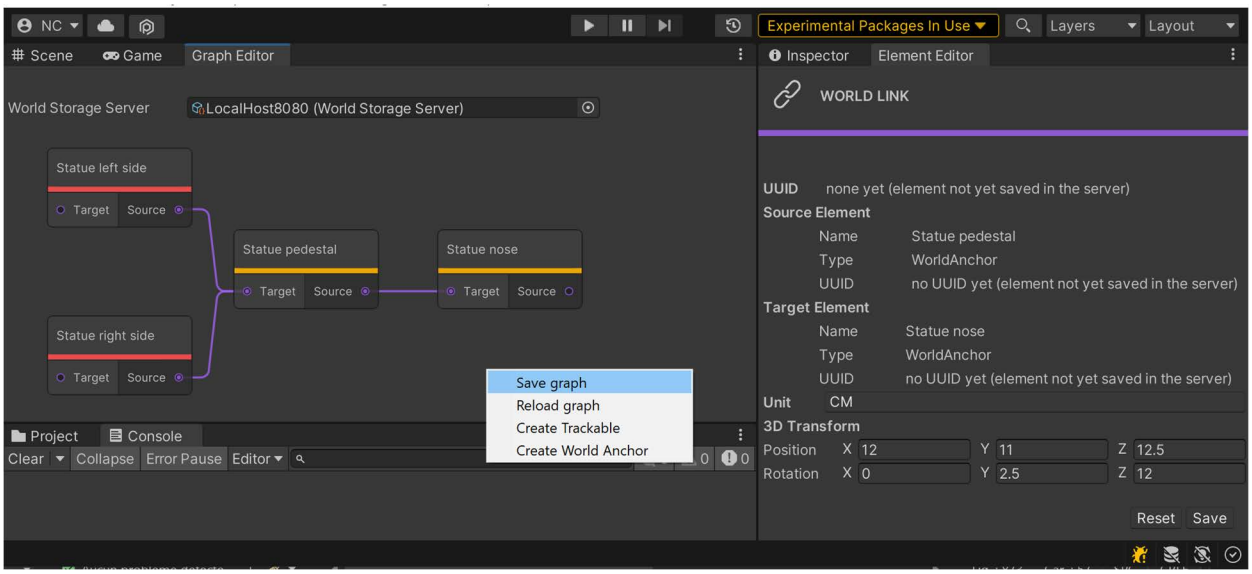


Figure B.12: The World Graph of the museum use case

Saving World Storage

However, those changes are only done locally. To save them to the World Storage, the implementer may select the ***Save graph*** option from the contextual menu (right click). Once selected, the graph is saved to the World Storage and the implementer will be able to retrieve it from any tool that communicates with the server.

NOTE: The implementer also has the possibility to save a specific element to the server by clicking the button ***Save*** on its editor inspector window.

History

Document history		
V1.1.1	September 2022	Publication