COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

D7.6 – Data Transformation for 3D BIM Rendering v2

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310

This project has received funding from the Eu
innovation programme under grant agreement No 958310

D7.6 – Data Transformation for 3D BIM Rendering v2

Dissemination Level:	Public
Deliverable Type:	Demonstrator
Lead Partner:	Hypertech
Contributing Partners:	UCL, UPM
Due date:	31-10-2022
Actual submission date:	28-10-2022

Authors

Name	Beneficiary	Email
Vassiliadis, Michalis	Hypertech	m.vassiliadis@hypertech.gr
Tsalis, Christos	Hypertech	c.tsalis@hypertech.gr
Malavazos, Christos	Hypertech	c.malavazos@hypertech.gr
Giannakis, Giorgos	Hypertech	g.giannakis@hypertech.gr
Papafragkakis, Apostolos	Hypertech	a.papafragkakis@hypertech.gr
Kakarantzas, Pavlos	Hypertech	p.kakarantzas@hypertech.gr
Fotopoulou, Anastasia	Hypertech	a.fotopoulou@hypertech.gr
Charbi, Theofania	Hypertech	t.charbi@hypertech.gr
Karalis, Evangelos	Hypertech	e.karalis@hypertech.gr
Katsigarakis, Kyriakos	UCL	k.katsigrakis@ucl.ac.uk
Lilis, Georgios	UCL	g.lilis@ucl.ac.uk
Bernardos, Socorro	UPM	sbernardos@fi.upm.es
Gerpe, Salvador González	UPM	salvador.gonzalez.gerpe@upm.es
García-Castro, Raúl	UPM	rgarcia@fi.upm.es

Reviewers

Name	Beneficiary	Email
Moraitis, Panagiotis	QUE	p.moraitis@que-tech.gr
Straka, Martin	NT	straka@novitechgroup.sk

Version History

Version	Editors	Date	Comment
0.1	Hypertech	26.07.2022	Table of Contents updated
0.3	Hypertech, UCL	20.08.2022	Section 2 updated
0.5	Hypertech	10.10.2022	Sections 1 and 3 updated
0.6	Hypertech	21.10.2022	Draft version ready for internal review
0.7	QUE, NT	25.10.2022	First draft internal review
0.8	Hypertech	27.10.2022	Final version
1.0	Hypertech	28.10.2022	Submission to the EC portal

Disclaimer

©COGITO Consortium Partners. All right reserved. COGITO is a HORIZON2020 Project supported by the European Commission under Grant Agreement No. 958310. The document is proprietary of the COGITO consortium members. No copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights. The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Communities. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use, which may be made, of the information contained therein.

Executive Summary

Within COGITO, several Graphical User Interfaces (GUIs) are being deployed to interact with the construction project stakeholders. Composed of various supporting services of different TRLs and maturity levels, COGITO components that are being developed and delivered in WP4, WP5 and WP6 are supported by dedicated front-end solutions for the end-user enabling them to (i) navigate the as-planned and as-built input and output data as well as (ii) trigger actions whenever required. Despite the fact that the COGITO consortium comprises partners who bring existing GUIs that are adapted, configured and re-purposed for the needs of the project, only a subset of COGITO's GUIs is being developed from scratch to support the DigiTAR, Digital Command Centre (DCC) and VirtualSafety applications.

For that subset, targeting to a common look and feel end-user experience, within COGITO, the Digital Twin Platform, backbone of the COGITO ecosystem, is enhanced with the COGITO Unity Library, a collection of C# scripts that is delivered in the form of a Unity package. The COGITO Unity Library constitutes the main outcome of task "T7.3 – Data Transformation for 3D BIM Rendering".

The first version of the library was delivered in M18 and documented in the interim version of this document, titled "D7.5 – Data Transformation for 3D BIM Rendering v1". Accommodating all the refinements that were identified and implemented from M19 to M24 of the project to support the advances in the Digital Twin Planform, the subject of the present deliverable is to document the second release of the COGITO Unity Library. The necessary updates were realised through various integration with the Digital Twin Platform tests that have been performed during the aforementioned timeframe.

In summary, the library provides functionalities for establishing the connection with, discovering projects and querying their as-planned and as-built data from the Digital Twin Platform, performing the 3D Building Information Model to mesh-based data transformation, and consequently converting the 3D geometric representation of the BIM model into manageable formats for Unity integration. At this point it is worth mentioning that although the COGITO Unity Library was initially planned to support the 3D BIM rendering in Unity environments only, the contributing partners opted to deliver a full package that provides all the necessary functionalities for interfacing with the DTP, querying the as-planned, IoT location tracking, as well as the Digital Twin applications (quality control, work progress and health & safety) results and visualising them in any Unity application. As a proof of concept, the COGITO Unity Library has driven the development of the Digital Command Centre, a Unity webGL application, documented in "D7.8 – Construction Digital Twin 3D Visualisation module v2".

Table of contents

List of Figures

List of Tables

List of Acronyms

Term	Description
API	Application Programming Interface
AR	Augmented Reality
COGITO	Construction Phase diGItal Twin mOdel
D	Deliverable
DCC	Digital Command Centre
DigiTAR	Digital Twin visualisation with Augmented Reality
DoA	Description of Action
DTP	Digital Twin Platform
GUI	Graphical User Interface
H&S	Health and Safety
HTTPS	Hypertext Transfer Protocol Secure
QC	Quality Control
IFC	Industry Foundation Classes
IoT	Internet of Things
MTL	Waterfront Material Template Library
MVP	Minimum Viable Product
OBJ	Waterfront Object file
T	Task
VR	Virtual Reality
XML	Extensible Markup Language
WODM	Work Order Definition and Management

1 Introduction

1.1 Scope and Objectives of the Deliverable

Within COGITO, the vast majority of the planning and construction phases' digital twin applications, being developed and delivered in WP4, WP5 and WP6, are accompanied by Graphical User Interfaces (GUIs). Such GUIs enable the construction stakeholders to navigate the as-planned and as-built data of the construction project, as well as view and analyse the results of dedicated services for construction workflow planning, progress estimation and optimisation, erected elements' quality control, crew safety planning and hazards prevention. A subset of COGITO's digital twin applications GUIs is built to support 3D BIM visualisation, namely the Digital Twin visualisation with Augmented Reality (DigiTAR), the Digital Command Centre (DCC) and the VirtualSafety.

As stated in "D2.5 – COGITO system architecture v2", the DCC is being developed to be the off-site data visualisation solution aiming to support the project managers on monitoring the construction progress, resources location, detected quality control defects, safety measures and detected hazardous areas through visualisation and navigation of the 5D as planned-data, construction resources data and other data and annotations generated by the QC, H&S and Workflow digital twin services. The DCC is the main output of "T7.4 – 3D Mesh Data Quality and Consistency Checker and 3D Data Transformation Testing". The DigiTAR is being prototyped as a software package for commercial Augmented Reality (AR) head mounted displays, developed and delivered in "T5.4 – User Interface for Construction Quality Control", and it aims to provide to the end users on-site visualisation of geometric and visual quality control results (defects) as well as safety hazards using AR/mobile device. The VirtualSafety, developed within "T4.4 – Personalised Safety Education and Learning", aims to be a Virtual Reality (VR) application designed for Health and Safety educational and training purposes.

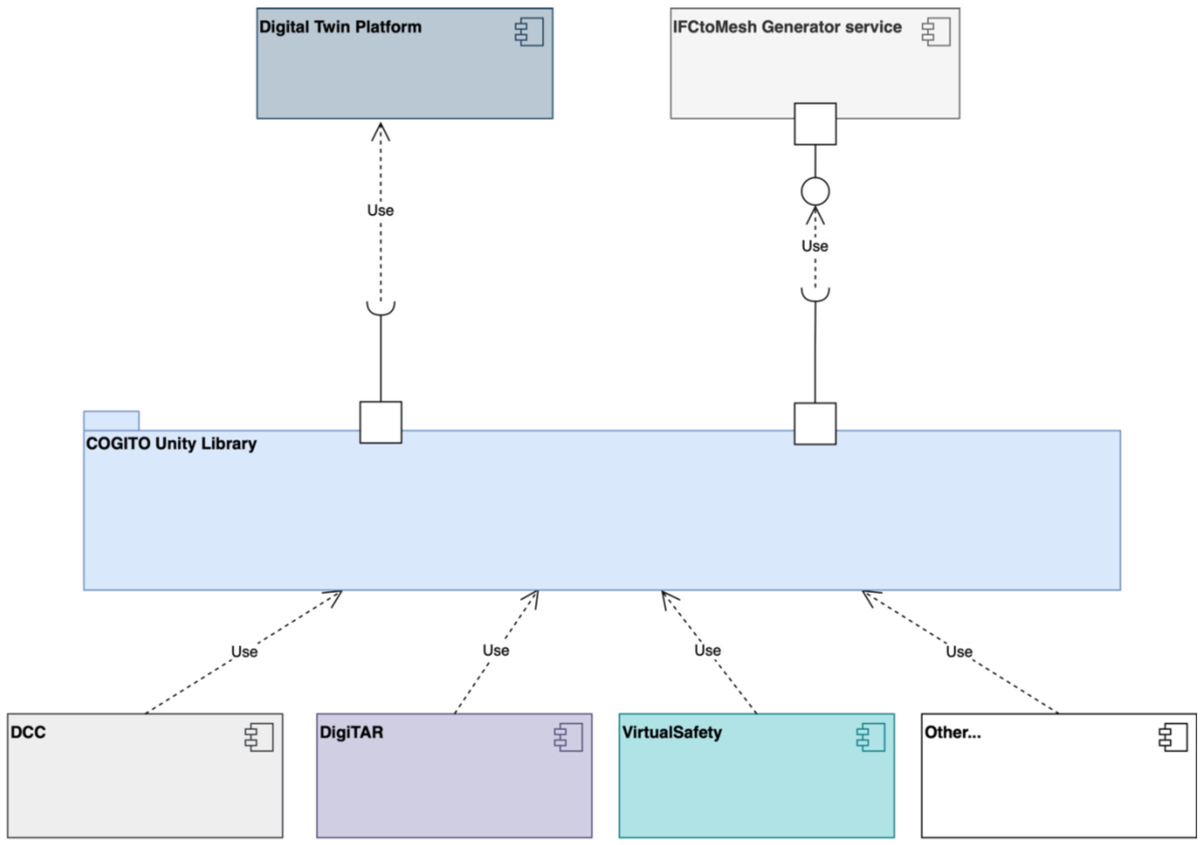


Figure 1 – The COGITO Unity Library can be used by any application that needs to communicate with the DTP and/or the IFCtoMesh Generator service

Common ground of the abovementioned applications constitutes the main framework that they build upon, namely the Unity Game Engine. Taking that into consideration and targeting to a unified end-user experience, a COGITO Unity Library is being developed. The COGITO Unity Library is a collection of C# scripts, delivered as a Unity package, that can be partly or fully integrated in any Unity application (see Figure 1). It is the outcome of task "T7.3-Data Transformation for 3D BIM Rendering", and the present document reports on the results of work that has been carried out towards delivering the first version of the library. The library provides methods to: (i) obtain the IFC file and convert it to a mesh-type file with accompanying metadata; (ii) enable the inter-communication of the IFCtoMesh Generator service with the DCC Unity application (or any other Unity-based application); (iii) establish the communication with the Digital Twin Platform (DTP) for obtaining real-time[1] data; (iv) obtain the mesh and metadata files from the IFCtoMesh Generator service and generate a Unity scene; (v) communicate with the DCC back-end; (vi) allow project selection; (vii) convert and render the data in the 3D view while allowing the user to select which data s/he wants to visualise; (viii) display a tree-like view of the building and allowing the user to select, filter and manipulate the elements while visualising element details and properties; and (ix) create a 3D visualisation of the as-planned and as-built elements, the IoT location data pre-processed by the IoT Data Pre-processing module, and the results of the Digital Twin applications (quality control, work progress and health & safety), stored in the DTP.

The COGITO Unity Library is developed in alignment with COGITO DoA to firstly support the design, development and testing of the DCC functionality in parallel with the development of the other components of the COGITO solution and prior to the commencement of the integration phase of the project, and secondly enable the development of a complete stand-alone component (DCC) that could be used as an innovative 3D viewer of any application (similar to COGITO) without any requirements for other components / services to be in place for ensuring its independent and interoperable operation.

1.2 Relation to other Tasks and Deliverables

Table 1 depicts the relations of this document to other deliverables within the COGITO project; the latter should be considered along with this document for further understanding of its contents.

Table 1 – Relation to other COGITO project's deliverables

Del. Number	Deliverable Title	Relations and Contribution
D2.1	Stakeholder requirements for the COGITO system	Analysis of the end-user requirements to create the necessary inputs for defining the COGITO Unity Library in conjunction with the DCC and its interactions with other components; a thorough description of the business scenarios, use cases and system requirements tailored to the project's goals setting the basis for the COGITO Unity Library development.
D2.5	COGITO System Architecture v2	The second version of the COGITO architecture report that includes updates on the specifications of the COGITO Unity Library in conjunction with the DCC (hardware/software requirements, programming language(s), development status, functional/non-functional requirements, inputs/outputs, along with the format, method, endpoint and protocol for each data type and interface).
D3.4	COGITO Data Model & Ontology Definition and Interoperability Design v3	Documentation of the third version of the COGITO ontologies (available online in the COGITO ontology portal).
D7.2	Digital Twin Platform Design & Interface Specification v2	Report on the Digital Twin Platform's detailed architecture, providing details on its authentication (identity and access management), data ingestion (project and BIM management), data

[1] Real-time data: IoT location data, quality control, work progress and health & safety results are often called hereafter real-time data

		persistence (file storage, project database), data management and messaging layers, required to establish the COGITO Unity Library communication with the DTP.
D7.8	Construction Digital Twin 3D Visualisation module v2	Second version of the Digital Command Centre – DCC, a Unity WebGL application that utilises the COGITO Unity Library and constitutes the off-site data visualisation solution, that helps the Project Manager to monitor through visualisation the construction progress, detected QC defects and H&S issues.
D7.10	Digital Twin Platform V2	Documentation of the second version of the DTP that provides instructions on how to authenticate users and applications, send and respond to data requests performed by COGITO applications.
D8.1	Integrated COGITO system V1	Documentation of the preliminary unit testing, packaging, and deployment activities of the individual tools along with the integration testing activities performed in the various use cases.

1.3 Structure of the Deliverable

To address the aspects relevant to the scope of T7.3, Section 1 introduces the work performed and the scope of this deliverable, along with its relevance to other COGITO tasks/deliverables and refinements that were performed in the COGITO Unity Library from M19 to M24 of the project to support the advances in the Digital Twin Planform.

Section 2 summarises the technical work that has been conducted up to M24 within T7.3 to deliver the first and the second version of the COGITO Unity Library. In a nutshell, Section 2.1 constitutes the core of this document, describing the architecture of the COGITO Unity Library, decomposing it into its sub-components and introducing their functionalities. The Inputs, Outputs and Application Programming Interfaces (APIs) of the library are introduced in Section 2.2. Section 2.3 presents the technology stack and implementation tools that the COGITO Unity Library developers have utilised to design and deliver it. Installation instructions on how to import the library in Unity projects, targeting to developers of COGITO Unity applications, are provided in Section 2.4. A thorough usage walkthrough of the COGITO Unity Library and its functionalities in Unity projects is presented in Section 2.5. Section 2 concludes with (i) the stakeholders, functional and non-functional requirements coverage analysis, (ii) the development and integration status, and (iii) the assumptions and restrictions of the second version of the library, reported in Sections 2.6, 2.7 and 2.8, respectively.

Finally, Section 3 presents the conclusions along with updates that are anticipated in the preparation of demonstration activities phase to guarantee a fully functional COGITO Unity Library and the IFCtoMesh Generator service ready to support the demonstration on the pilot sites.

1.4 Updates to the first version of the COGITO Unity Library

The focus of this deliverable has been to report on refinements and additions to the functionalities of the COGITO Unity Library, and relevant activities, that have already been documented in D7.5, based on verification and validation experiments performed in relevant environments (e.g., the school sample project). Hence, this document complements D7.6, providing information about actions that have been taken to address the assumptions and restrictions that have been reported in D7.5, namely the development of modules that facilitate the communication and data exchange with the DTP. Thus, contrasting the 2nd with the 1st version of the COGITO Unity Library document the following modifications and additions have been performed: (1) Section 2.1.4.1 has been added to describe the COGITO Unity Library's authentication flow through the Identity Provided offered by the DTP; (2) all classes descriptions provided throughout Section 2.1 have been updated to reflect their current status – the updates were realised and implemented during various data exchange tests with the DTP; finally, (3) Sections 2.1, 2.4 and 2.5 have been updated accordingly to present results based on actual data that have been retrieved via calls to the DTP endpoints.

2 COGITO Unity Library for Digital Twin data visualisation

2.1 Overall Architecture

The COGITO Unity Library is a collection of C# scripts and is delivered in the form of a Unity package. It uses the Unity API to achieve its objectives. Figure 2 illustrates the overall architecture of the library and its interaction with other components, realised during its integration with the DCC. As stated above, even though Figure 2 presents the COGITO Unity Library as part of the DCC application, the library is a reusable package that can be fully or partly utilised by any COGITO-specific Unity application (e.g., DigiTAR and VirtualSafety).

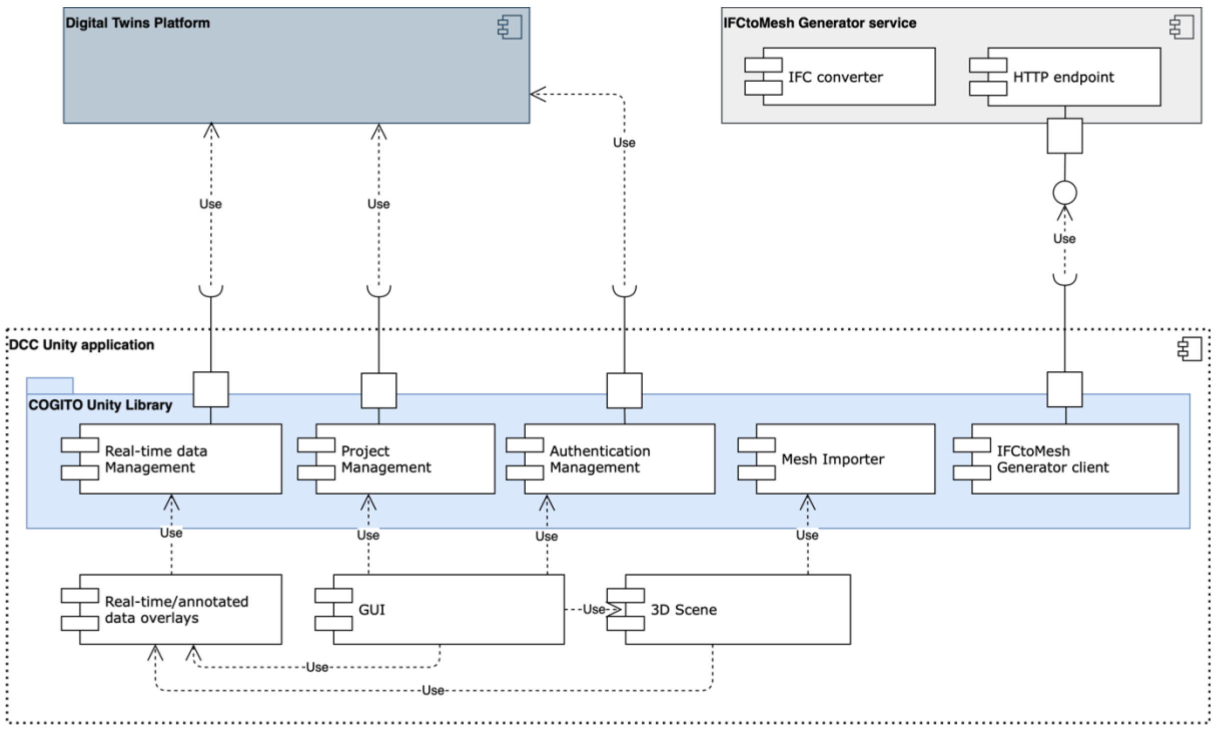

Digital Twins Platform
IFCtoMesh Generator service
IFC converter
HTTP endpoint
Use
DCC Unity application
COGITO Unity Library
Real-time data Management
Project Management
Authentication Management
Mesh Importer
IFCtoMesh Generator client
Real-time/annotated data overlays
GUI
3D Scene

Figure 2 – Overview of the COGITO Unity library as part of the DCC (D7.8) application

In summary, the COGITO Unity Library is composed of five main components (from right to left in Figure 2) that are further described in the following sections:

- the **IFCtoMesh Generator client** that invokes the IFCtoMesh Generator service to convert an IFC file to geometry, metadata, and material files;
- the **Mesh Importer** component providing the functionalities required to import geometry, metadata, and material files and -if needed - convert IFC files to geometry and material files;
- the **Authentication Management** component component implements the OpenID Connect logic supported by the DTP Keycloak service;
- the **Project Management** component, responsible for communicating with the DTP and obtaining information about the available projects and the project data files; and
- the **Real-time data Management** component, that regularly sends request to the DTP to receive real-time data related to Health & Safety, Quality Control, Workflow and IoT location tracking data based on a predefined scheduler.

Although not part of the COGITO Unity Library, the **IFCtoMesh Generator** is an external framework that serves as a data transformation service that receives as input IFC files and generates geometry (OBJ), metadata (XML), and material (MTL) files that can be used by clients upon request.

2.1.1 IFCtoMesh Generator service

The IFCtoMesh Generator is a standalone service that processes and transforms IFC data to mesh-based geometry and metadata files. The service is based on the IfcOpenShell open-source project. In IfcOpenShell, the OpenCASCADE (the Open CASCADE Community Edition) geometry kernel is used to perform the geometric operations, required to transform the implicit geometry in IFC files into explicit geometry that can be imported into Unity. In IfcOpenShell, the developers maintain two components that are of particular interest for the IFCtoMesh Generator: (i) the IfcConvert, an IFC to mesh-based geometry and metadata converter; and (ii) the ifc-pipeline, a web-based IFC viewer that utilizes IfcConvert.

The IfcConvert component is an executable tool that accepts IFC files as input and produces geometry files and metadata files. IfcConvert supports the following output file types:

- WaveFront OBJ;
- Collada - Digital Assets Exchange;
- glTF - Binary glTF v2.0;
- STEP - Standard for the Exchange of Product Data;
- IGES - Initial Graphics Exchange Specification;
- XML - Property definitions and decomposition tree;
- SVG - Scalable Vector Graphics (2D floor plan); and
- IFC-SPF - Industry Foundation Classes.

Unity does not provide any APIs that allow importing of geometry on runtime. Given that fact, the OBJ has been selected, since it is the easiest format to be parsed and imported.

The ifc-pipeline component is a processing queue that uses IfcOpenShell to convert IFC input files into a graphic display using glTF 2.0 and BIMSurfer2 for visualization. The project is comprised of two parts (see Figure 3):

- the Server-side back-end that receives IFC files and converts them; and
- the Web-based front-end, a small viewer for IFC files.

As part of the IFCtoMesh Generator service, the backend of the ifc-pipeline project is only uitilised, since within COGITO, custom Unity BIM viewer are being developed. Note here part of our experimentation with the ifc-pipeline has been the ifc-pipeline backend update to be capable of generating OBJ instead of GLB files.

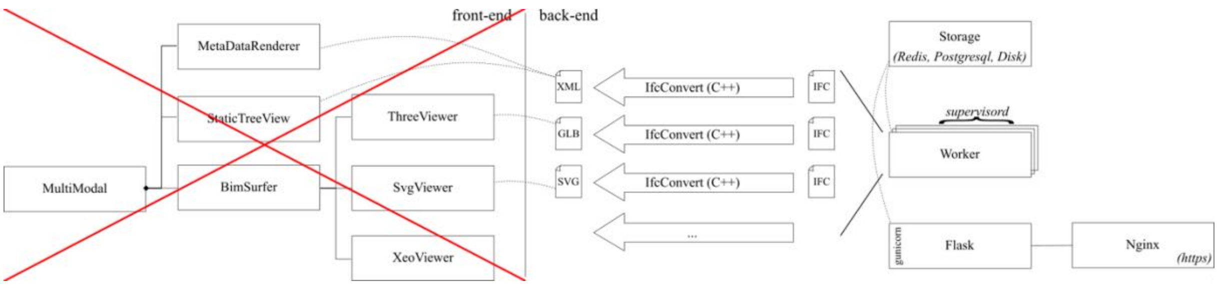


Figure 3 – ifc-pipeline block diagram with the front-end part deleted[2]

2.1.2 IFCtoMesh Generator client

The IFCtoMesh Generator client provides the means to interact with the IFCtoMesh Generator service from any Unity application developed within COGITO. It resides in the `COGITOUnityPackage.IfcToMeshGeneratorClient` namespace. Its main exposed method is the `ConvertAndDownload`. The method accepts a stream object that contains the IFC file data and proceeds with the following actions:

1. upload the file to the IFCtoMesh Generator service using an HTTPS POST;

[2] https://github.com/AECgeeks/ifc-pipeline

2. if successful, start monitoring the conversion progress by polling requests to the endpoint of the IFCtoMesh Generator service (the polling interval is 100ms);
3. once the progress is 100%, start downloading the conversion result which is comprised of the following files: (i) OBJ file containing geometry; (ii) XML file containing IFC metadata and hierarchy; and (iii) MTL file containing material properties.

The file data can be accessed from the IFCtoMesh Generator client instance using the following public properties: `streamObj`, `streamXml`, and `streamMtl`.

The IFCtoMesh Generator client also contains properties that can be used to monitor the conversion status from the application: (i) `ConversionProgress` that represents the current conversion progress as reported by the IFCtoMesh Generator service; and (ii) `IsLoading`, a Boolean indicating whether the client is currently in the process of converting and/or loading files. The `ConversionProgressResponse` class is used to store the conversion progress status from the IFCtoMesh Generator service. It contains a single property, `Progress`, which is of integer type and takes a value from 0 to 100. The `ConversionProgressResponse` class is instantiated and filled with data by deserializing the JSON responses from the relevant calls to the IFCtoMesh Generator service. The IFCtoMesh Generator client's classes are thoroughly described in Table 2 and Table 3.

Table 2 – COGITO Unity Library: IfcToMeshGeneratorClient class

`class IfcToMeshGeneratorClient`
The `IfcToMeshGeneratorClient` class contains all the functionality needed to interface with the IFCtoMesh Generator service in order to convert IFC files to OBJ, XML, MTL and As-Planned formats and import them to the current scene.
Namespace: `COGITOUnityPackage.IfcToMeshGeneratorClient`
Inherits: `MonoBehaviour`
Methods: - `public void ConvertAndDownload(Stream ifcStream) ->` it uploads an IFC file to the IFCtoMesh Generator service, starts the conversion, monitors the conversion and loads the resulting object to the currently active scene. Accepts a C# I/O stream object pointing to the IFC file data. The asynchronous nature of contacting an external service is handled with the Unity-native way of coroutines. - `public async Task<GameObject> LoadOfflineObject(String ifcPath) ->` a Utility method that loads an IFC file to the current scene by converting it locally using the IfcConvert executable that is in the project root directory. It accepts the file path of the IFC. It cannot be used in WebGL.
Properties: - `public string BackendUrl ->` it contains the URL of the deployed DccBackend used to connect to the service - `private string SessionId ->` it contains the currently running conversion session id, or `null` - `public bool IsLoading -> true` if a conversion is currently running - `public ConversionProgressResponse ConversionProgress ->` it contains the current progress of the conversion

Table 3 – COGITO Unity Library: ConversionProgressResponse class

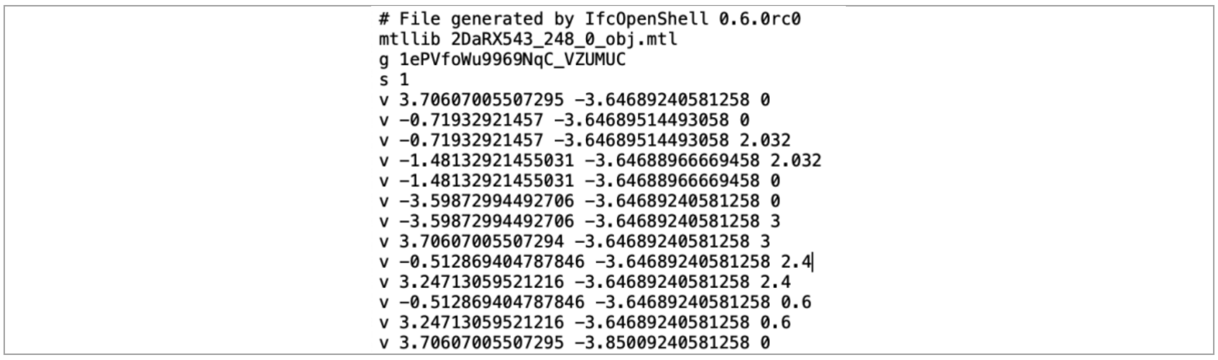
`class ConversionProgressResponse`
A Utility class that contains the progress of the currently running conversion. Can be used by a Unity project to show a UI progress bar for the conversion.
Namespace: -
Inherits: none
Methods: none
Properties: `public int Progress ->` A number from 0 to 100 indicating the progress of the currently running conversion

2.1.3 Mesh Importer

OBJ files contain geometric information only and do not support any other metadata or additional information on the objects. Since IFC files contain a multitude of extra data on the building elements, IfcConvert also generates an XML file with all the IFC metadata, linked with the generated meshes that reside in the OBJ file. Given these 2 files (OBJ and XML), the COGITO Unity Library can create GameObjects that contain geometry, hierarchy, and metadata. Furthermore, IfcConvert generates a MTL (material) file that contains the properties of the used materials. Since textures are not captured in IFC, the MTL file contains information about the colour and opacity of the generated meshes. To accommodate the as-planned construction schedule data of the building elements (4th dimension of BIM), the IFC files are enriched with relevant information: custom property sets named `Construction` that contain an ID that is referenced in a separate file that is downloaded from the DTP along with the IFC.

2.1.3.1 Wavefront OBJ

OBJ is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer[3] animation package. The file format is open and has been adopted by other 3D graphics application vendors as well. The OBJ file format is a simple data-format that represents 3D geometry alone — namely, the position of each vertex, the uv position[4] of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices[5] (see Figure 4).



```
# File generated by IfcOpenShell 0.6.0rc0
mtllib 2DaRX543_248_0_obj.mtl
g 1ePVfoWu9969NqC_VZUMUC
s 1
v 3.70607005507295 -3.64689240581258 0
v -0.71932921457 -3.64689514493058 0
v -0.71932921457 -3.64689514493058 2.032
v -1.48132921455031 -3.64688966669458 2.032
v -1.48132921455031 -3.64688966669458 0
v -3.59872994492706 -3.64689240581258 0
v -3.59872994492706 -3.64689240581258 3
v 3.70607005507294 -3.64689240581258 3
v -0.512869404787846 -3.64689240581258 2.4
v 3.24713059521216 -3.64689240581258 2.4
v -0.512869404787846 -3.64689240581258 0.6
v 3.24713059521216 -3.64689240581258 0.6
v 3.70607005507295 -3.85009240581258 0
```

Figure 4 – Subset of an OBJ file generated by IfcConvert

The OBJ format can also contain named groups of vertices/faces/polygons which helps with the linking of the IFC element metadata with the generated polygon. The "g" line in Figure 2 shows such a group definition.

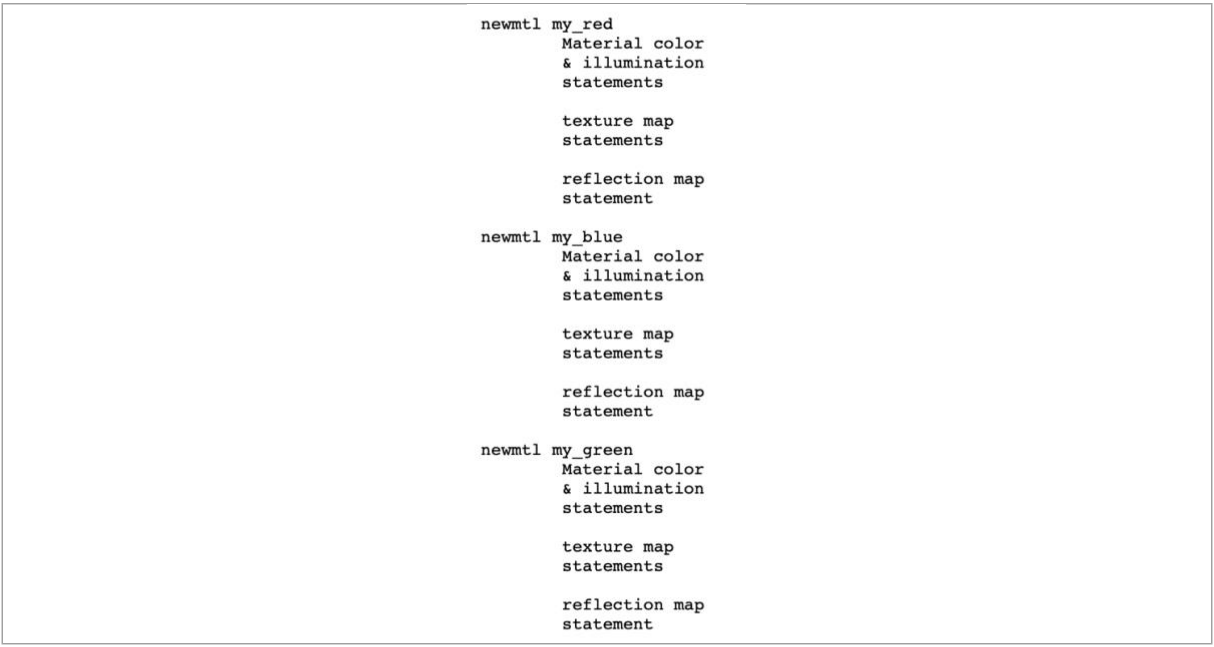
[3] https://en.wikipedia.org/wiki/The_Advanced_Visualizer
[4] The letters "u" and "v" denote the axes of the 2D texture.
[5] https://en.wikipedia.org/wiki/Wavefront_.obj_file

IfcConvert by default uses an internal group ID but using the `use-element-guids` option allows us to use the element IDs from the IFC.

2.1.3.2 Material Library File (MTL)

Material library files contain one or more material definitions, each of which includes the color, texture, and reflection map of individual materials. These are applied to the surfaces and vertices of objects. Material files are stored in ASCII format and have the .MTL extension.

An MTL file is typically organized as shown in Figure 5.



```
newmtl my_red
        Material color
        & illumination
        statements

        texture map
        statements

        reflection map
        statement

newmtl my_blue
        Material color
        & illumination
        statements

        texture map
        statements

        reflection map
        statement

newmtl my_green
        Material color
        & illumination
        statements

        texture map
        statements

        reflection map
        statement
```

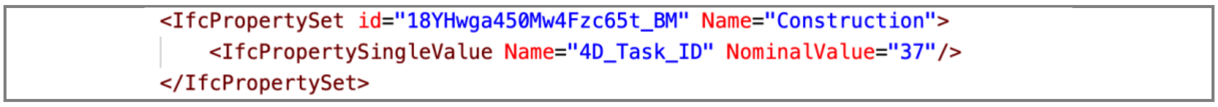
Figure 5 – Typical organization of an .MTL file

2.1.3.3 Metadata XML file

IfcConvert has the capability to generate an XML file that accompanies the OBJ file. This file is required since the OBJ file captures the mesh properties (vertices, faces, 15ormal) only without any metadata that describe the relations between elements as they are captured in IFCs.

The XML file's data is similar to the IFC (with some extra simplifications) but formatted as XML. Having an XML file instead of an IFC makes it much easier to parse (code-wise). The XML file consists of the following sub-sections: Header; Units; Properties; Quantities; Types; Layers; Materials; and Decomposition.

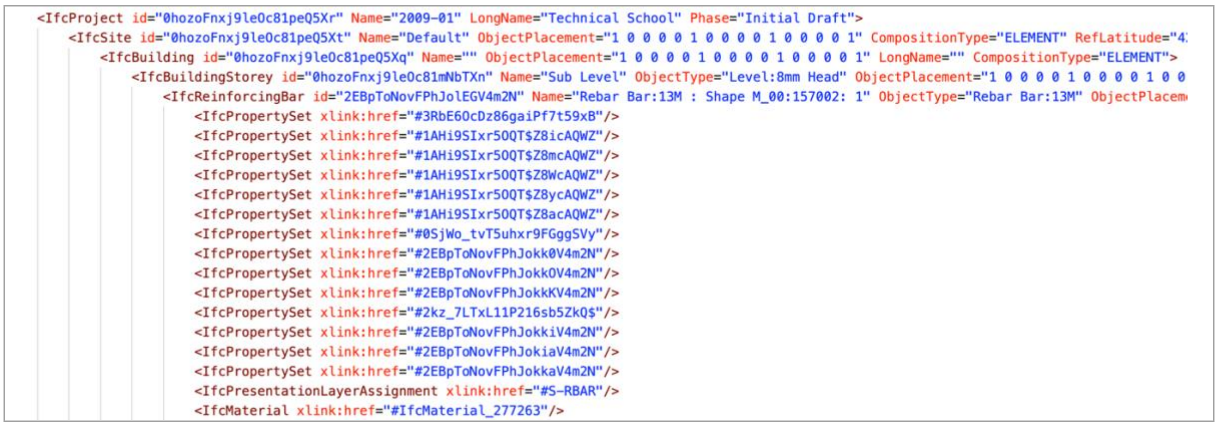
The Properties and Decomposition sections are of particular interest. The *Properties* section references all building elements and define their property sets. An example of the "Construction" property set definition in XML format is presented in Figure 6.



```
<IfcPropertySet id="18YHwga450Mw4Fzc65t_BM" Name="Construction">
    <IfcPropertySingleValue Name="4D_Task_ID" NominalValue="37"/>
</IfcPropertySet>
```

Figure 6 – Example of the "Construction" property set definition in XML format

The *Decomposition* section defines the actual hierarchy of the elements along with references to property sets (defined above), units and other metadata coming from the IFC (see Figure 7).



```
<IfcProject id="0hozoFnxj9le0c81peQ5Xr" Name="2009-01" LongName="Technical School" Phase="Initial Draft">
    <IfcSite id="0hozoFnxj9le0c81peQ5Xt" Name="Default" ObjectPlacement="1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1" CompositionType="ELEMENT" RefLatitude="4
        <IfcBuilding id="0hozoFnxj9le0c81peQ5Xq" Name="" ObjectPlacement="1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1" LongName="" CompositionType="ELEMENT">
            <IfcBuildingStorey id="0hozoFnxj9le0c81mNbTXn" Name="Sub Level" ObjectType="Level:8mm Head" ObjectPlacement="1 0 0 0 0 1 0 0 0 0 1 0 0
                <IfcReinforcingBar id="2EBpToNovFPhJolEGV4m2N" Name="Rebar Bar:13M : Shape M_00:157002: 1" ObjectType="Rebar Bar:13M" ObjectPlacem
                    <IfcPropertySet xlink:href="#3RbE6OcDz86gaiPf7t59xB"/>
                    <IfcPropertySet xlink:href="#1AHi9SIxr50QT$Z8icAQWZ"/>
                    <IfcPropertySet xlink:href="#1AHi9SIxr50QT$Z8mcAQWZ"/>
                    <IfcPropertySet xlink:href="#1AHi9SIxr50QT$Z8WcAQWZ"/>
                    <IfcPropertySet xlink:href="#1AHi9SIxr50QT$Z8ycAQWZ"/>
                    <IfcPropertySet xlink:href="#1AHi9SIxr50QT$Z8acAQWZ"/>
                    <IfcPropertySet xlink:href="#0SjWo_tvT5uhxr9FGggSVy"/>
                    <IfcPropertySet xlink:href="#2EBpToNovFPhJokk0V4m2N"/>
                    <IfcPropertySet xlink:href="#2EBpToNovFPhJokkOV4m2N"/>
                    <IfcPropertySet xlink:href="#2EBpToNovFPhJokkKV4m2N"/>
                    <IfcPropertySet xlink:href="#2kz_7LTxL11P216sb5ZkQ$"/>
                    <IfcPropertySet xlink:href="#2EBpToNovFPhJokkiV4m2N"/>
                    <IfcPropertySet xlink:href="#2EBpToNovFPhJokiaV4m2N"/>
                    <IfcPropertySet xlink:href="#2EBpToNovFPhJokkaV4m2N"/>
                    <IfcPresentationLayerAssignment xlink:href="#S-RBAR"/>
                    <IfcMaterial xlink:href="#IfcMaterial_277263"/>
```

Figure 7 – Start of the Decomposition section in XML

2.1.4 Authentication Management

The Authentication Management component implements the OpenID Connect logic supported by the DTP Keycloak service. It allows the configuration of the application ID and secret and the execution of the authentication flow to obtain a session to the DTP. The DTP session is represented as a token that is included in all subsequent HTTPS messages to the DTP.

All other COGITO Unity Library components that communicate with the DTP use the token Authentication Management component to access the token and include it in the HTTP message headers as needed. The Authentication Management, as well as an authenticated session is required for all operations, except the IFCtoMesh Generator service operations, since the IFCtoMesh Generator service is independent from the DTP and can be used by itself.

The Authentication Management provides the following features (see the AuthenticationManagement class description in Table 4 for further details):

- a way to input the application ID and secret;
- the Authenticate method, called to create a DTP session; and
- the authentication token used by all other components (except IfcToMeshGeneratorClient) to access the DTP API.

Table 4 – COGITO Unity Library: AuthenticationManagement class

`class AuthenticationManagement`
The `AuthenticationManagement` implements the DTP OpenID Connect authentication flows. It provides the Authentication method which – if successful – provides an authentication session to the DTP via a token to be used by all other COGITO Unity Library components that communicate with the DTP.
Namespace: `COGITOUnityPackage.AuthenticationManagement`
Inherits: `MonoBehaviour`
Methods: - `public IEnumerator Authenticate(string username, string password, Action<bool> onAuthenticationFinished) ->` it authenticates to the DTP using the provided username and password. Throws an exception if not successful. The `onAuthenticationFinished` parameter is a call-back that will be called when the process is finished with a parameter of bool type, indicated whether the authentication was successful.

- `public bool IsLoggedIn() ->` it checks if an authenticated session is available. Returns true if authenticated, false if no token is available.

Properties:
- `public string applicationId ->` it contains the – Keycloak provided – application ID
- `public string applicationSecret ->` it contains the – Keycloak provided – application secret
- `public bool token ->` it contains the token obtained from Keycloak after a successful authentication
- `public string username ->` it contains the username used to obtain the session. If no session is available contains null.

2.1.4.1 Authentication flow
Since the authentication endpoint of DTP and the actual DTP provided services are on different servers, the authentication flow involves communicating with three different endpoints, as illustrated in Figure 8:

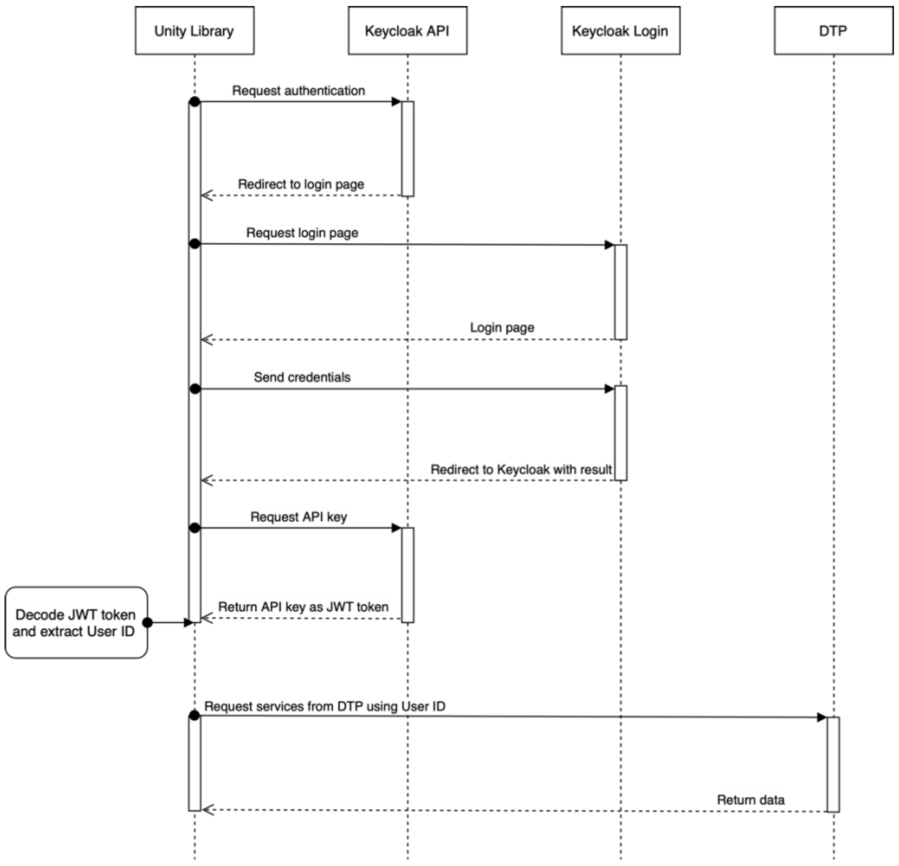
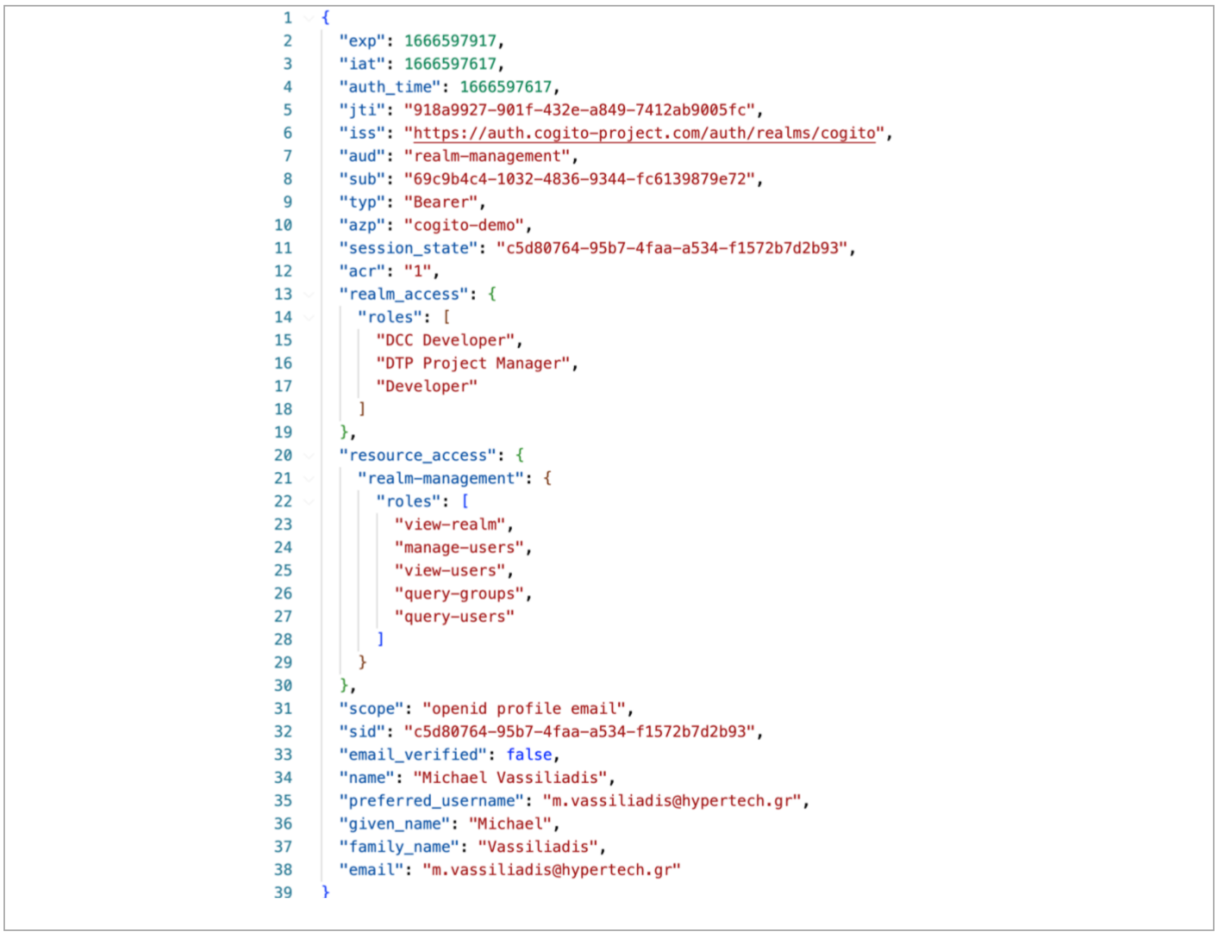


Figure 8 – COGITO services authentication flow

- **Unity Library**: The Unity Library initiating the authentication flow;
- **Keycloak API**: The HTTP API provided by Keycloak to accommodate the delegated authentication process;
- **Keycloak Login**: The Keycloak login page that checks the user provided credentials;
- **DTP**: The COGITO DTP plarform providing the authenticated services.

The authentication process results in an API key that is in JWT format. JSON Web Token (JWT) is an open standard, defined by RFC 7519 that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.



```
{
  "exp": 1666597917,
  "iat": 1666597617,
  "auth_time": 1666597617,
  "jti": "918a9927-901f-432e-a849-7412ab9005fc",
  "iss": "https://auth.cogito-project.com/auth/realms/cogito",
  "aud": "realm-management",
  "sub": "69c9b4c4-1032-4836-9344-fc6139879e72",
  "typ": "Bearer",
  "azp": "cogito-demo",
  "session_state": "c5d80764-95b7-4faa-a534-f1572b7d2b93",
  "acr": "1",
  "realm_access": {
    "roles": [
      "DCC Developer",
      "DTP Project Manager",
      "Developer"
    ]
  },
  "resource_access": {
    "realm-management": {
      "roles": [
        "view-realm",
        "manage-users",
        "view-users",
        "query-groups",
        "query-users"
      ]
    }
  },
  "scope": "openid profile email",
  "sid": "c5d80764-95b7-4faa-a534-f1572b7d2b93",
  "email_verified": false,
  "name": "Michael Vassiliadis",
  "preferred_username": "m.vassiliadis@hypertech.gr",
  "given_name": "Michael",
  "family_name": "Vassiliadis",
  "email": "m.vassiliadis@hypertech.gr"
}
```

Figure 9 – Example of decoded JWT from Keycloak

For subsequent calls to DTP services the User ID contained in the returned token is required, appearing as the “sub” parameter in the JWT.

2.1.5 DTP Endpoint and Data Collections system

The COGITO Digital Twins Platform uses a hierarchical system for communicating with external components as well as process and distribute data from/to all interested components. This system allows the DTP to separate data and processing into areas of concern and handle each input and output in a compartmentalized way to achieve the following: each external component has access only to the data it needs; each input is processed as needed using the internal DTP graphs; and all data input/output operations are asynchronous.

The “elements” used within the DTP hierarchy are the following (see Figure 10): (1) Applications; (2) Endpoints; and (3) Data Collections. Each application that needs to communicate with the DTP is assigned an endpoint that is created by the DTP administrator for a specific data exchange purpose, for example Health and Safety data. Each endpoint can contain collections, which are file repositories. Collections can be created either by the application that owns the parent endpoint, or by DTP itself. The two cases are described below:

- An application creates an endpoint to send data to the DTP. The application creates a collection, uploads files and marks the collection as completed.
- The DTP needs to send data to an application. Upon marking the data collection completed in the above step, the DTP might run predefined processes on the collected files. These processes might include the creation of data collections of other endpoints and their population with files as needed.

In the case of the IFC file, the DTP creates a collection in the application's endpoint and puts the IFC file in it. The Unity Library communicates with the DTP API to identify the collection and download the IFC file from it.

The DTP API provides the following collection relevant calls: (1) get all endpoints of an application; (2) get all collections of an endpoint; (3) get collection; (4) upload files into a collection; (5) get all files of a collection. For each data collection need, the endpoints are created as a configuration step of the application and the UUID of the endpoint exists as a fixed configuration parameter within the Unity Library. The endpoint for the IFC in our example is `51d1540c-2762-4668-8964-14464bc53a8c`.

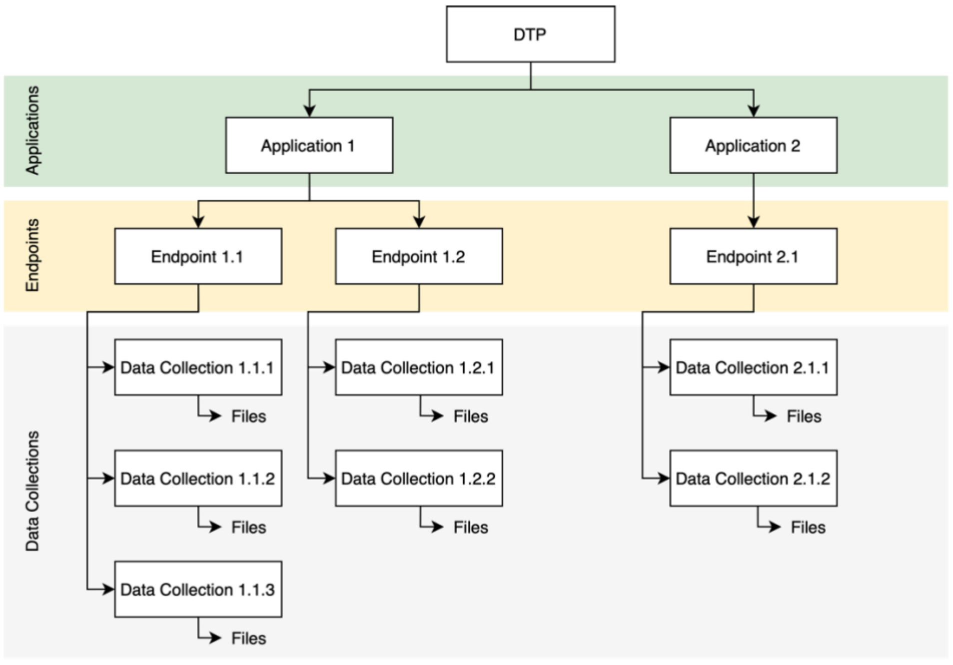


Figure 10 – DTP Applications, Endpoints and Data Collections hierarchy

2.1.5.1 Endpoints predefined for the Unity Library

The following endpoints are defined within the DTP endpoint, along with their UUIDs:

1. DCC Input 1 – 4D BIM data - `51d1540c-2762-4668-8964-14464bc53a8c;`
2. DCC Input 2 – Resources tracking - `082f82a0-4585-43ff-94ba-694a75a7515f;`
3. DCC Input 3 – Quality control results - `588a0ede-7752-4e3c-9fce-58a7aca56692;`
4. DCC Input 4 – Health & Safety results - `1b9a3b8a-d96b-4bf3-b747-4c4e16253ce7;`
5. DCC Input 5 – Workflow progress - `d7256071-9e5b-4799-b993-9e0d56fe6171.`

Each of these endpoints contains at least one collection, which provides the JSON file with the respective data. Having more than one collections is supported and allows the Unity Library to obtain historical data.

The data contained in each data collection as well as its usage for the purposes of the Unity Library will be described in the relevant sections below.

2.1.6 Project Management

The Project Management component implements functionalities to fetch the list of projects from the DTP, select a project, and retrieve the relevant project files. The project list contains the projects that are accessible by the logged in user. The Project Management component resides in the `COGITOUnityPackage.ProjectManagement` namespace and provides two methods:

- the `GetProjectList` method that contacts the DTP and requests the available projects – when the resulting list is obtained the `results` callback is called, supplied with the list of projects; and

- the `GetProjectIfc` method that receives the project id (as supplied by the `GetProjectList` method) and a callback that accepts a stream parameter - when the IFC file has been downloaded the callback is invoked with the stream object containing the file data as parameter.

The corresponding class is presented in Table 5. The Project Management component also provides the `ProjectInfo` class that contains the retrieved project information.

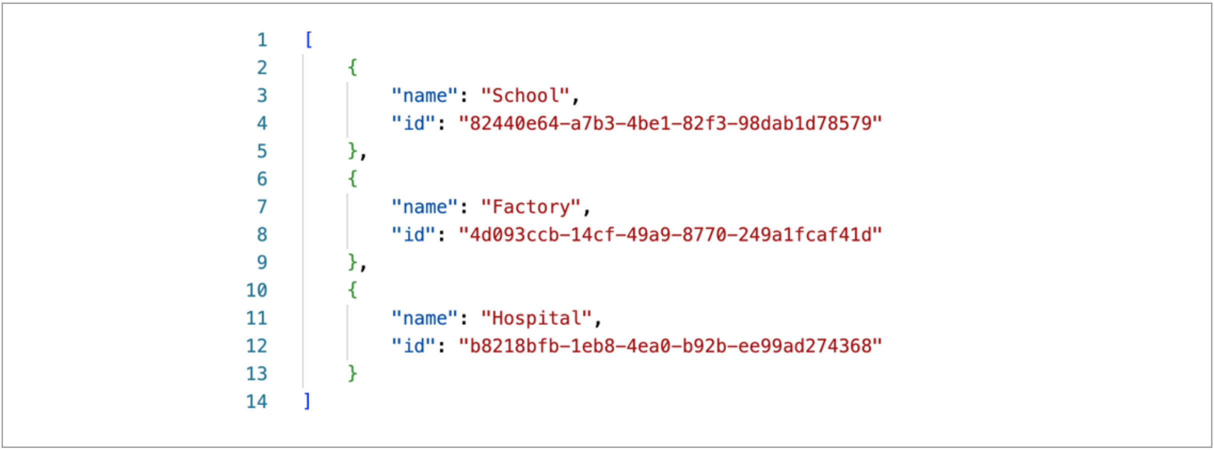
Table 5 – COGITO Unity Library: ProjectManagement class

`class ProjectManagement`
The `ProjectManagement` class contains all the functionality needed to contact the DTP and get the list of available projects as well as the IFC file for a particular project.
Namespace: `COGITOUnityPackage.ProjectManagement`
Inherits: `MonoBehaviour`
Methods: - `public IEnumerator GetProjectList(Action<List<ProjectInfo>> results) ->` it contacts the DTP to obtain the list of projects available to the currently logged in user. This should be executed as a coroutine i.e., using `StartCoroutine`. The `results` parameter is a user-defined callback that will be called with the resulting list of projects as a parameter. - `public IEnumerator GetProjectIfc(int projectId, Action<Stream> stream)` -> it contacts the DTP to obtain the IFC of a particular project. The currently logged in user should have access to the project, otherwise an exception will be thrown. This should be executed as a coroutine i.e., using `StartCoroutine`. The `projectId` parameter is the project ID as returned from the `GetProjectList` method. The `stream` parameter is a user-defined callback that will be called with the resulting IFC stream data. - `public IEnumerator GetAsPlannedDataForElement(int projectId, string elementId, Action<Tuple<DateTime, DateTime>> asPlanned)` -> Contacts the DTP to obtain the As-Planned data (start date, end date) for the given project and element. The `asPlanned` tuple contains the start and end dates for the element.
Properties: - `public List<ProjectInfo> ProjectList ->` it contains the projects returned from the DTP - `public string ProjectManagementUrl ->` it contains the URL of the Project Management endpoint of the DTP - `public Guid 4dBimEndpointId ->` it contains the 4D BIM data endpoint UUID - `public Guid WodmEndpointId ->` it contains the Work Progress data endpoint UUID - `public Guid ResourceTrackingEndpointId ->` it contains the resources tracking data endpoint UUID - `public Guid QualityControlEndpointId ->` it contains the quality control data endpoint UUID - `public Guid HealthAndSafetyEndpointId ->` it contains the Health & Safety data endpoint UUID The above properties need to be set by the user of the library to the - assigned by the DTP administrator - UUIDs of the endpoints for the particular application. - `public Guid SelectedProject ->` The user sets this to the Guid of the project the application is currenlty working on.

2.1.6.1 DTP Project enumeration

The DTP API provides the following project related calls: (1) get all projects; (2) get project; (3) get all users of a project; (4) get all properties of a project; and (5) get all projects of a user. The project management API of the Unity Library uses the "`get all projects of a user`" call. This accepts the user ID (obtained by the authentication flow) and returns a JSON list of all the projects the user has access to, along with their UUIDs. The HTTP request that generated the following example was:

https://dtp.cogito-project.com/api/users/69c9b4c4-1032-4836-9344-fc6139879e72/projects



```
[
    {
        "name": "School",
        "id": "82440e64-a7b3-4be1-82f3-98dab1d78579"
    },
    {
        "name": "Factory",
        "id": "4d093ccb-14cf-49a9-8770-249a1fcaf41d"
    },
    {
        "name": "Hospital",
        "id": "b8218bfb-1eb8-4ea0-b92b-ee99ad274368"
    }
]
```

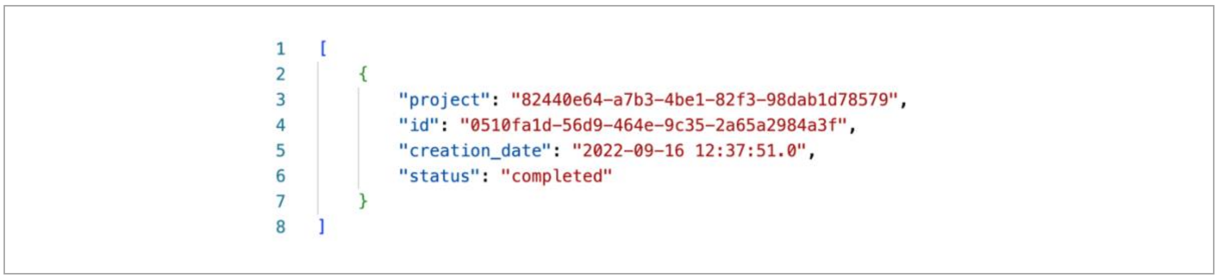
Figure 11 – DTP to COGITO Unity Library: Example of project list JSON

2.1.6.2 Retrieval of the project IFC

To get the available data collections within the endpoint, the "`get all collections of an endpoint`" call is used:

https://dtp.cogito-project.com/api/endpoints/51d1540c-2762-4668-8964-14464bc53a8c/collections

The above call results in a JSON with the following content:



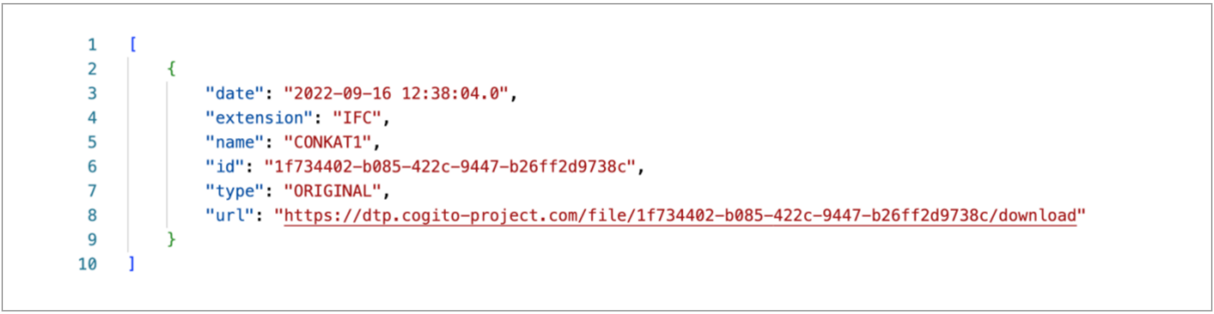
```
[
    {
        "project": "82440e64-a7b3-4be1-82f3-98dab1d78579",
        "id": "0510fa1d-56d9-464e-9c35-2a65a2984a3f",
        "creation_date": "2022-09-16 12:37:51.0",
        "status": "completed"
    }
]
```

Figure 12 – DTP to COGITO Unity Library: Data collection within the 4D BIM endpoint

The Unity Library verifies that the status of the collection is "completed" and proceeds to locate and download the IFC file using the "`get all files of a collection`" call:

https://dtp.cogito-project.com/api/collections/0510fa1d-56d9-464e-9c35-2a65a2984a3f/files

Which results in the following JSON:



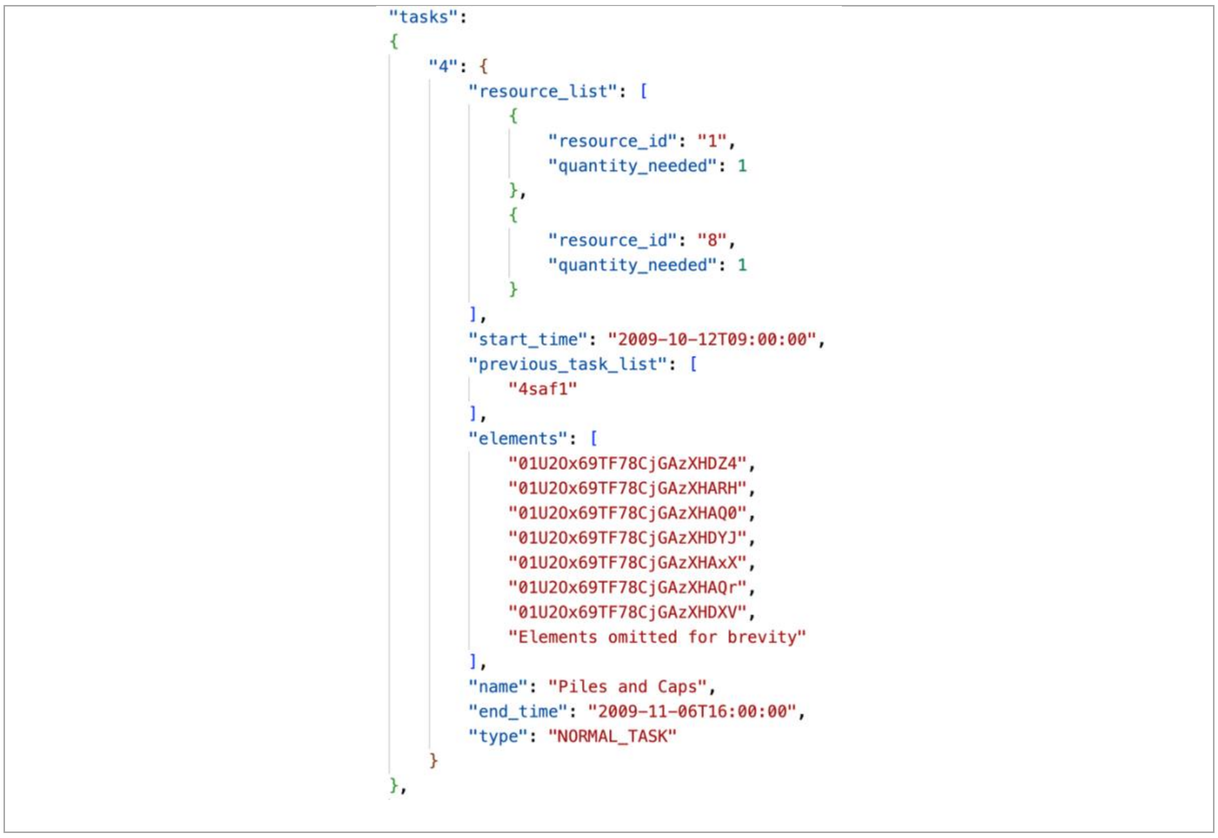
```
[
    {
        "date": "2022-09-16 12:38:04.0",
        "extension": "IFC",
        "name": "CONKAT1",
        "id": "1f734402-b085-422c-9447-b26ff2d9738c",
        "type": "ORIGINAL",
        "url": "https://dtp.cogito-project.com/file/1f734402-b085-422c-9447-b26ff2d9738c/download"
    }
]
```

Figure 13 – DTP to COGITO Unity Library: IFC file within the data collection

The Unity Library then proceeds to download the file indicated by the "`url`" parameter.

2.1.6.3 As-Planned schedule data

The “4D BIM” endpoint contains a second data collection with the As-Planned schedule data. The JSON file has a top-level property named “tasks”, which is a dictionary of tasks. Each task contains various properties about the described task including the start date, the end date and the BIM elements involved.



```
"tasks":
{
    "4": {
        "resource_list": [
            {
                "resource_id": "1",
                "quantity_needed": 1
            },
            {
                "resource_id": "8",
                "quantity_needed": 1
            }
        ],
        "start_time": "2009-10-12T09:00:00",
        "previous_task_list": [
            "4saf1"
        ],
        "elements": [
            "01U20x69TF78CjGAzXHDZ4",
            "01U20x69TF78CjGAzXHARH",
            "01U20x69TF78CjGAzXHAQ0",
            "01U20x69TF78CjGAzXHDYJ",
            "01U20x69TF78CjGAzXHAxX",
            "01U20x69TF78CjGAzXHAQr",
            "01U20x69TF78CjGAzXHDXV",
            "Elements omitted for brevity"
        ],
        "name": "Piles and Caps",
        "end_time": "2009-11-06T16:00:00",
        "type": "NORMAL_TASK"
    }
},
```

Figure 14 – DTP to COGITO Unity Library: As-Planned schedule data for a task involving elements

2.1.7 Real-Time Data Management

The Real-Time Data Management component operates independently after the initial data import step (i.e., the import of OBJ, XML, MTL and As-Planned schedule data). It can refresh the data any time the application needs to be updated, or even in regular intervals. The following data groups are supported:

- Health & Safety (H&S);
- Quality Control (QC);
- Workflow Management (WODM); and
- Asset & Resource Location Tracking (IoT).

The library obtains the data on demand. Whenever the application requires an update/refresh of the presented/currently available data, a request to the DTP is sent. The data can subsequently be used to present overlays, textual information, or any other way the developer sees fit. The Real-Time Data Management classes are described in Tables 6-11.

Table 6 – COGITO Unity Library: RealTimeDataManagement class

`class RealTimeDataManagement`
The `RealTimeDataManagement` class contains all the functionality needed to contact with and get the real-time data available in the DTP. The supported data types are: - Health & Safety (H&S) applications results; - Quality Control (QC) applications results; - Workflow Management (WODM) results; and - Resources (equipment and workers) Location Tracking (IoT) data.

Namespace:
`COGITOUnityPackage.RealTimeDataManagement`

Inherits:
`MonoBehaviour`

Methods:

- `public async List<QualityControlResult> GetQualityControlResultsForElement(string elementId) ->` Fetches the quality control results for a particular element. All Quality Control entries that involve the specified element are returned in a list.
- `public async List<QualityControlResult> GetQualityControlResults() ->` Fetches all quality control results for the project.
- `public async List<Resource> GetAllResources() ->` Fetches all resouces for the current project.
- `public async List<Resource> GetResourcesByNameOrTag(string name, Guid tag) ->` Fetches resources by name or tag.
- `public void SubscribeToResourceTracking(Action<ResourceLocation> OnDataReceived) ->` Subscribes to the MQTT broker in order to receive real-time coordinate tracking data for resources. Throws "Unable to subscribe" exception.
- `public void UnsubscribeFromResourceTracking()` -> Stops receiving resource tracking updates from the MQTT broker. Throws "Unable to unsubscribe" exception.
- `public async WorkflowProgress GetProgressForElement(string elementId) ->` Gets the progress data for a particular element id.
- `public async Dictionary<string, WorkflowProgress> GetProgressForAll() ->` Gets the progress data for all elements in the current project.

Table 7 – COGITO Unity Library: QualityControlResult class

`class QualityControlResult`
The `QualityControlResult` class contains an entry from the Quality Control system.
Properties: - `string Description ->` Textual description of the result - `DateTime ScheduleDate` -> Scheduled date of check - `DateTime PerformedDate ->` Actual date of check - `string Units ->` Unit in measurement - `double ScalarResult ->` Result of measurement - `double ToleranceReference`: `->` Pass/Fail threshold for measurement - `bool Pass ->` TRUE if check is a Pass - `string[] InvolvedComponents ->` Array of element IDs involved in the check

Table 8 – COGITO Unity Library: ResourceType enumeration

`enum ResourceType`
Contains the type of a resource

Values: - HUMAN - EQUIPMENT

Table 9 – COGITO Unity Library: Resource class

`class Resource`
The Resource class contains information about a resource (human or equipment)
Properties: - `string Name ->` Name of the resource - `ResourceType Type ->` Type of the resource (human or equipment) - `Guid Tag ->` Tracking tag of the resource - `string FirstName ->` First name of the resource (if human) - `string LastName ->` Last name of the resource (if human) - `string Email ->` Email of the resource (if human) - `string[] Roles ->` Array of roles the resource has (if human) - `Guid KeycloakId ->` Keycloak UUID of the resource (if human)

Table 10 – COGITO Unity Library: ResourceLocation class

`class ResourceLocation`
The ResourceLocation class contains the coordinates of a resource (human or equipment)
Properties: - `Resource Resource:` Reference to the resource - `Vector3 Position:` Position coordinates of the resource - `DateTime Timestamp:` Timestamp of the location information

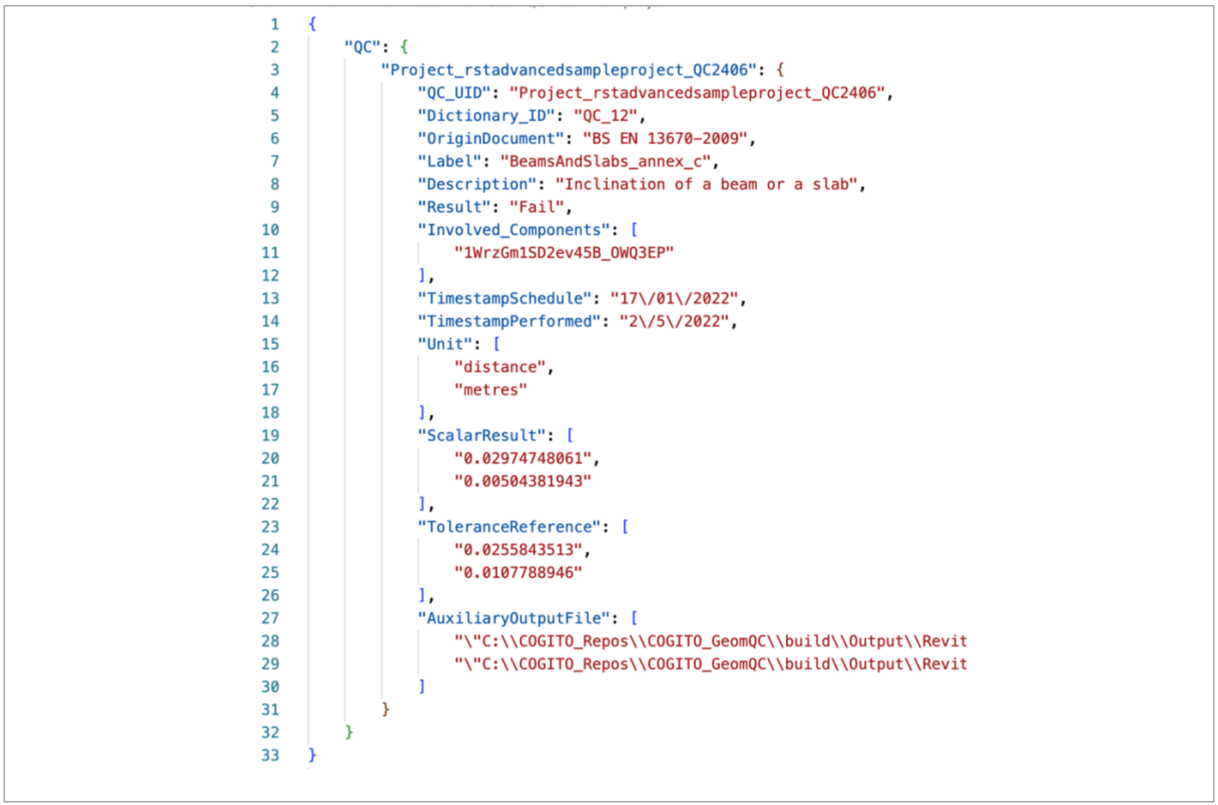
Table 11 – COGITO Unity Library: WorkflowProgress class

`class WorkflowProgress`
The WorkflowProgress class contains the construction progress of an element.
Properties: - `string ElementId:` Element ID - `double Progress:` progress (from 0 – not started, to 1 - completed)

2.1.7.1 Quality Control results data

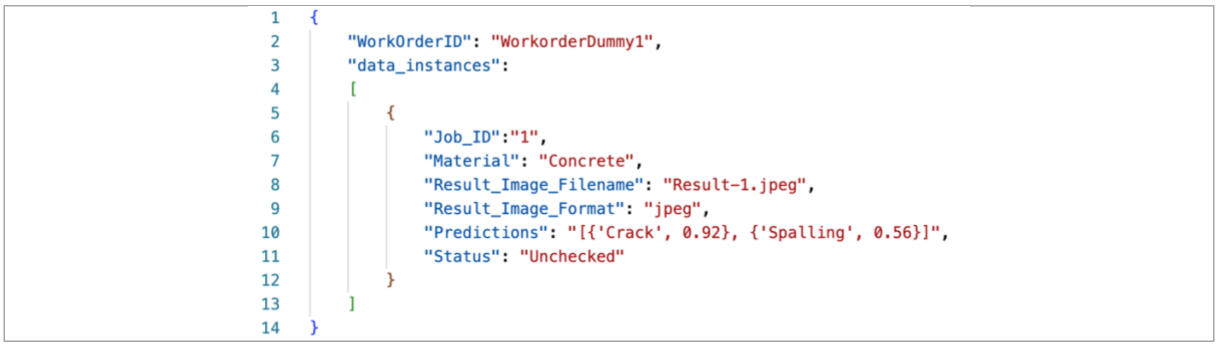
Quality control results come in two types: Geometric and Visual. The responsible endpoint for the Unity Library should contain two collections, one for each type.

Visual and Geometric quality control results have different parameters as shown below.



```json
{
    "QC": {
        "Project_rstadvancedsampleproject_QC2406": {
            "QC_UID": "Project_rstadvancedsampleproject_QC2406",
            "Dictionary_ID": "QC_12",
            "OriginDocument": "BS EN 13670-2009",
            "Label": "BeamsAndSlabs_annex_c",
            "Description": "Inclination of a beam or a slab",
            "Result": "Fail",
            "Involved_Components": [
                "1WrzGm1SD2ev45B_OWQ3EP"
            ],
            "TimestampSchedule": "17\/01\/2022",
            "TimestampPerformed": "2\/5\/2022",
            "Unit": [
                "distance",
                "metres"
            ],
            "ScalarResult": [
                "0.02974748061",
                "0.00504381943"
            ],
            "ToleranceReference": [
                "0.0255843513",
                "0.0107788946"
            ],
            "AuxiliaryOutputFile": [
                "\"C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\Revit
                "\"C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\Revit
            ]
        }
    }
}
```

Figure 15 – DTP to COGITO Unity Library: Geometric Quality control data



```json
{
    "WorkOrderID": "WorkorderDummy1",
    "data_instances":
    [
        {
            "Job_ID":"1",
            "Material": "Concrete",
            "Result_Image_Filename": "Result-1.jpeg",
            "Result_Image_Format": "jpeg",
            "Predictions": "[{'Crack', 0.92}, {'Spalling', 0.56}]",
            "Status": "Unchecked"
        }
    ]
}
```

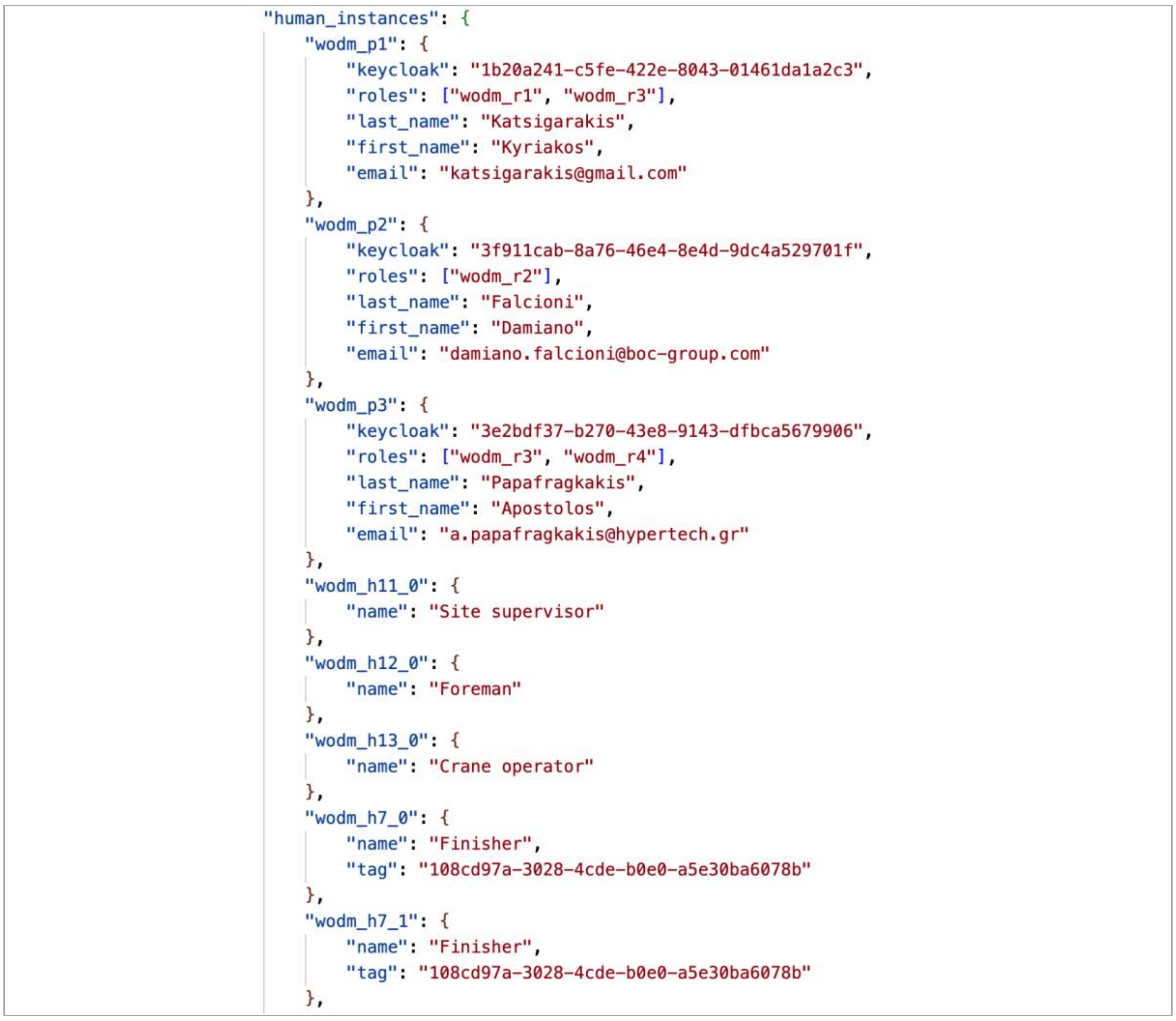
Figure 16 – DTP to COGITO Unity Library: Visual Quality control data

2.1.7.2 Health & Safety results and Work Progress data

In alignment with the latest updates to the DTP and the maturity of the Health & Safety (ProactiveSafety) and Workflow Management (Work Order Definition and Management – WODM and Work Order Execution Assistance – WOEA) Digital Twin applications of COGITO in the construction phase, the real-time health & safety and work progress data formats undergo refinements. Currently, such refinements are under evaluation before being integrated into the COGITO data model and ontology. Thus, relevant data were not available to be queried using the "DCC Input 4 – Health & Safety results" and the "DCC Input 5 – Workflow progress" endpoints (see Section 2.1.5.1). Hence, they remain subject for testing to be performed in T8.1.

2.1.7.3 Resource Tracking data

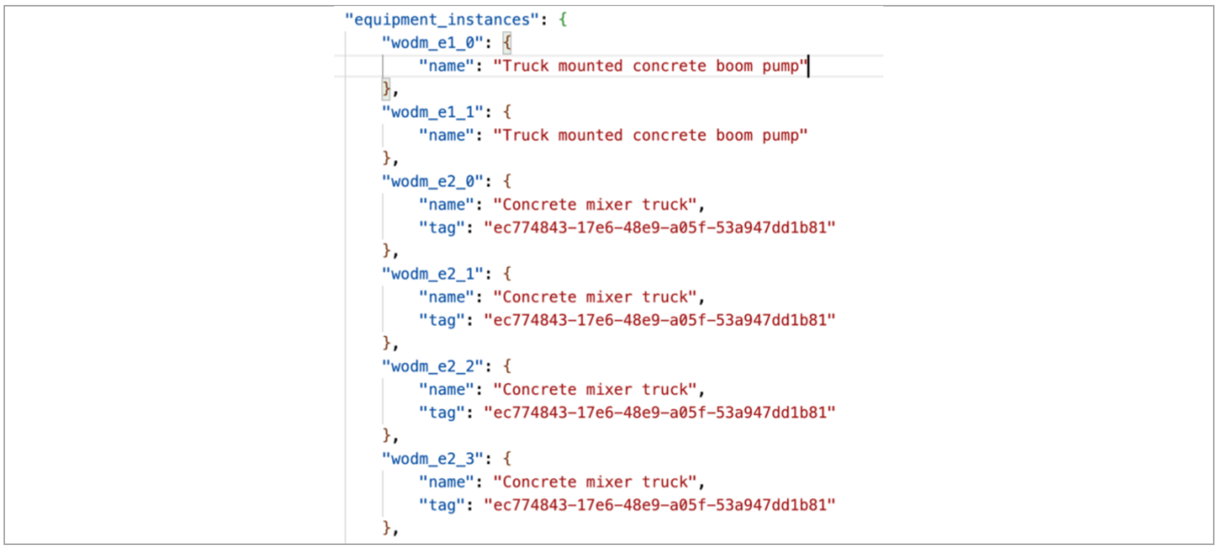
Resource tracking data retrieval through the DTP is achieved using a hybrid method: (1) historical data by REST APIs calls to DTP endpoints; and (2) real-time data by subscription to a DTP message queue. Using the endpoint "DCC Input 2 – Resource Tracking" (see Section 2.1.5.1) the library can obtain a JSON file that contains information about all resources used in the project in two separate lists, one for equipment and one for humans. Each entry in these lists includes the name of the resource and its UUID tracking tag.



```
"human_instances": {
    "wodm_p1": {
        "keycloak": "1b20a241-c5fe-422e-8043-01461da1a2c3",
        "roles": ["wodm_r1", "wodm_r3"],
        "last_name": "Katsigarakis",
        "first_name": "Kyriakos",
        "email": "katsigarakis@gmail.com"
    },
    "wodm_p2": {
        "keycloak": "3f911cab-8a76-46e4-8e4d-9dc4a529701f",
        "roles": ["wodm_r2"],
        "last_name": "Falcioni",
        "first_name": "Damiano",
        "email": "damiano.falcioni@boc-group.com"
    },
    "wodm_p3": {
        "keycloak": "3e2bdf37-b270-43e8-9143-dfbca5679906",
        "roles": ["wodm_r3", "wodm_r4"],
        "last_name": "Papafragkakis",
        "first_name": "Apostolos",
        "email": "a.papafragkakis@hypertech.gr"
    },
    "wodm_h11_0": {
        "name": "Site supervisor"
    },
    "wodm_h12_0": {
        "name": "Foreman"
    },
    "wodm_h13_0": {
        "name": "Crane operator"
    },
    "wodm_h7_0": {
        "name": "Finisher",
        "tag": "108cd97a-3028-4cde-b0e0-a5e30ba6078b"
    },
    "wodm_h7_1": {
        "name": "Finisher",
        "tag": "108cd97a-3028-4cde-b0e0-a5e30ba6078b"
    },
```

Figure 17 – DTP to COGITO Unity Library: Instances of human resources

Human resources can also contain further identification properties such as their first and last name, their email and their roles and UUID within the project (as defined in Keycloak). This information appears only on human resources that are applicable.



```
"equipment_instances": {
    "wodm_e1_0": {
        "name": "Truck mounted concrete boom pump"
    },
    "wodm_e1_1": {
        "name": "Truck mounted concrete boom pump"
    },
    "wodm_e2_0": {
        "name": "Concrete mixer truck",
        "tag": "ec774843-17e6-48e9-a05f-53a947dd1b81"
    },
    "wodm_e2_1": {
        "name": "Concrete mixer truck",
        "tag": "ec774843-17e6-48e9-a05f-53a947dd1b81"
    },
    "wodm_e2_2": {
        "name": "Concrete mixer truck",
        "tag": "ec774843-17e6-48e9-a05f-53a947dd1b81"
    },
    "wodm_e2_3": {
        "name": "Concrete mixer truck",
        "tag": "ec774843-17e6-48e9-a05f-53a947dd1b81"
    },
```

Figure 18 – Instances of equipment resources

Equipment resources can only contain the name and a tracking tag (if tracked). Using the "tag" parameter of the resources the Unity Library can further retrieve coordinate data for the tracked resources from a MQTT broker. Humans and Equipment trackers publish coordinate data in specific MQTT topics and the information is received by the Unity Library in real time, which in turn passes the data to the callback specified in the `SubscribeToResourceTracking` method.

2.2 API Documentation

Table 8 presents the API methods exposed by the IFCtoMesh Generator service. The rest components do not expose any APIs; the interfaces with the DTP are achieved with the import of the COGITO Unity Library into a Unity application.

Table 8 – IFCtoMesh Generator service API methods

Path	Method	Description
/	POST	Upload the IFC file and immediately start the conversion; the worker <id> is returned
/pp/<id>	GET	Get the Conversion progress in JSON format
/m/<id>.<ext>	GET	Returns the requested file for download. Example: /m/123.obj or /m/123.xml
/log/<id>.<ext>	GET	Returns the conversion log. Example: /log/123.json

2.3 Technology Stack and Implementation Tools

The technologies and the implementation tools and libraries that are used by the COGITO Unity Library are summarised in Table 9.

Table 9 – Libraries and Technologies used in COGITO Unity Package

Library/Technology Name	Version	License
IfcOpenShell	0.6.0rc0	LGPL-3.0 License
Arcventure - IFC importer	1.2	Standard Unity Asset Store EULA - Extension Asset
Unity	2021.2.18f1	Proprietary
WebGL	2.0	Not Applicable

2.4 Installation Instructions

Unity custom packages are distributed using a single file that needs to be imported in a Unity project. The package files come with a `.unitypackage` extension. To use the COGITO Unity Library, the user needs to follow the steps below:

1. Obtain the COGITO Unity Library package file;
2. Start a new Unity project or open an existing one that needs to consume DTP or IFCtoMesh Generator service data;
3. From the main Unity menu select Assets → Import Package → Custom Package… (see Figure 19)

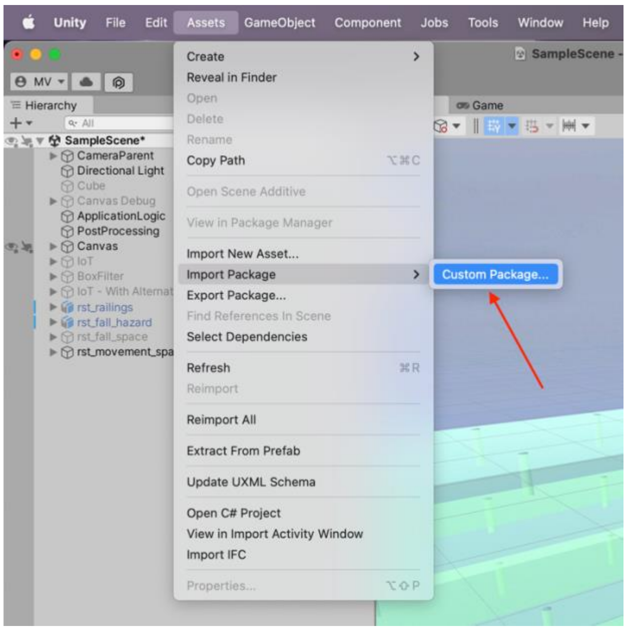


Figure 19 – Import custom Unity package to a Unity project

4. Unity will present a modal window allowing the user to choose the package file. Locate the file on your disk and select it. Then click on Open (see Figure 20)

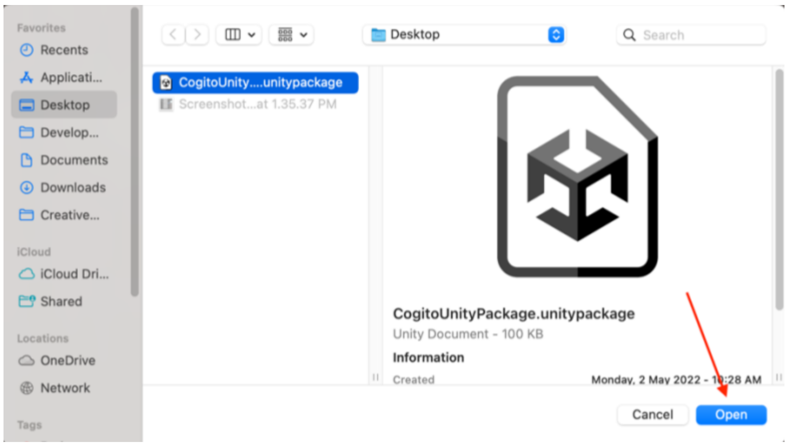


Figure 20 – Selection of package for import to Unity library

5. A window pops up where the user selects the files, contained in the package, that will be imported in the Unity project. Select all (preselected by default) and click on Import (see Figure 21).

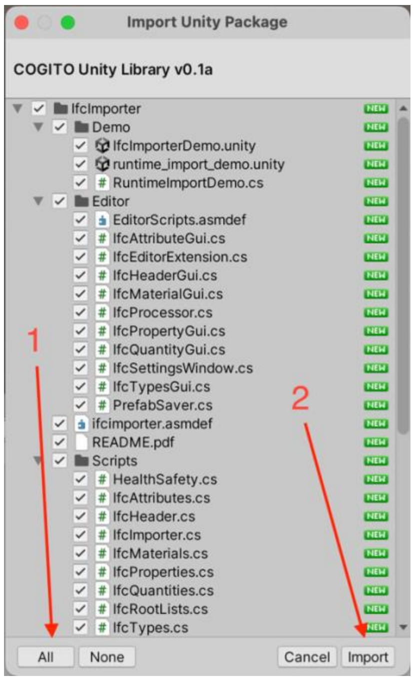


Figure 21 – Selection of the package assets to be included in the import

6. All package assets have been imported to the project resources as indicated in the resources window (see Figure 22).

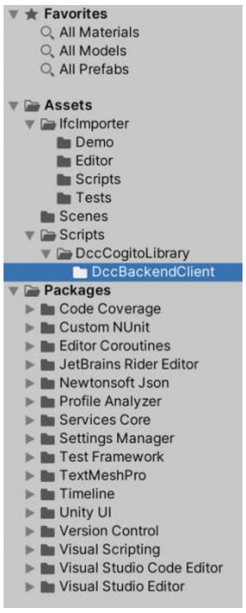


Figure 22 – Package with selected assets imported to the Unity project

The APIs and classes provided be the COGITO Unity Library are now ready to be used in the project.

2.5 Usage Walkthrough

After importing the COGITO Unity Library in the Unity project as described in Section 2.4, the following examples can be used as a starting point for a variety of use cases.

2.5.1 Library initialisation

To use the library, a GameObject that has the main library object must be created. The name of the GameObject is set by the Unity application's developer. For the examples that follow, we assume that the name of the GameObject is `COGITOGameObject`.

Create a new GameObject and attach the following scripts to it, as needed. The `AuthenticationManagement` (Authentication Manager in Figure 23) script is required for all cases since no communication can be established with the DTP before authenticating. An exception to this is the `IfcToMeshGeneratorClient` which does not depend on the DTP.

- `AuthenticationManagement` (Required)
- `IfcToMeshGeneratorClient`
- `ProjectManagement`
- `RealTimeDataManagement`

Some scripts need extra configuration which can be edited via the Unity editor. Instructions for these will be provided in the relevant sections below.

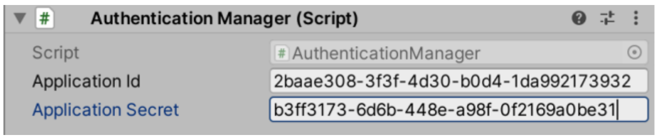
2.5.2 Authenticating to the DTP

To use any feature of the library the authentication process must first be executed. This requires the following information:

- Once for each application, the application id and secret must be entered. The id and secret are provided by the DTP administrators.
- A username and password must be provided by the user, either programmatically or from a UI within the app. The chosen method depends on the developed application requirements.

2.5.2.1 Providing the application id and secret

After selecting the GameObject that contains the library scripts from the Unity hierarchy panel, the following will appear in the inspector panel, among other things.


Authentication Manager (Script)

Script	AuthenticationManager
Application Id	2baae308-3f3f-4d30-b0d4-1da992173932
Application Secret	b3ff3173-6d6b-448e-a98f-0f2169a0be31

Figure 23 – Application ID and secret in the authentication manager

Enter the application ID and secret provided by the DTP administrators to the corresponding fields.

2.5.2.2 Authenticating a user to the server

The following line of code can be used to authenticate.

```
…
var authManager =
GameObject.Find("COGITOGameObject").GetComponent<AuthenticationManager>();
StartCoroutine(authManager.Authenticate("username", "password",
OnAuthenticationFinished));
…

private void OnAuthenticationFinished(bool success)
{
    Debug.Log($"Authentication was { success ? "successful" : "unsuccessful" }");
}
```

`Authenticate` will throw an exception if authentication fails. `authManager.IsLoggedIn()` can be used at any point to check if the session is authenticated.

2.5.3 Fetching projects

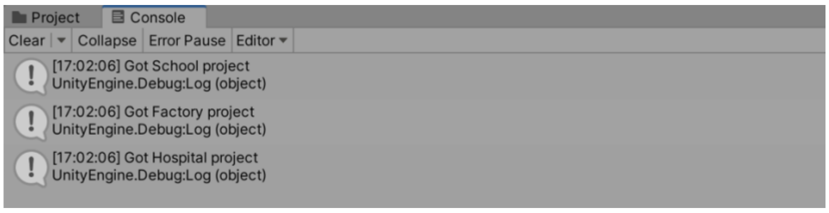
2.5.3.1 Getting the list of available projects

Use the following code to fetch the list of available projects for the current user.

```
…
var projectManagement =
GameObject.Find("COGITOGameObject").GetComponent<ProjectManagement>();
StartCoroutine(projectManagement.GetProjectList(OnProjectListReceived));
…

private void OnProjectListReceived(List<ProjectInfo> projects)
{
    foreach (var project in projects)
    {
        Debug.Log($"Got { project.Name } project");
    }
}
```

The above example will produce the following output:



```
[17:02:06] Got School project
UnityEngine.Debug:Log (object)
[17:02:06] Got Factory project
UnityEngine.Debug:Log (object)
[17:02:06] Got Hospital project
UnityEngine.Debug:Log (object)
```

Figure 24 – Output of GetProjectList example code

The projects information can be found in the projects `List`. Of special importance is the `project.Id` for each project that is needed to download the IFC and all other project-related information.

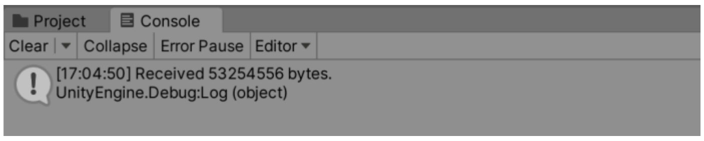
2.5.3.2 Getting a project IFC

Use the following code to obtain a project IFC file.

```
…
StartCoroutine(GetProjectIfc(project.Id, OnProjectStreamReady));
…

private void OnProjectStreamReady(Stream ifcStream)
{
    // Read the stream here
}
```

The above code produces the following output:



```
[17:04:50] Received 53254556 bytes.
UnityEngine.Debug:Log (object)
```

Figure 25 – Output of GetProjectIfc example code

The stream can be used in any way C# and the .NET framework allows.

2.5.4 Using the IFCtoMesh Generator service

The IFCtoMesh Generator service is used to convert and import BIM information from an IFC file into the currently active Unity scene.

```
…
```

```
var IfcToMeshGeneratorClient =
GameObject.Find("COGITOGameObject").GetComponent<IfcToMeshGeneratorClient>();
IfcToMeshGeneratorClient.ConvertAndDownload(ifcStream);
…
```

The above snippet will initiate a connection to the IFCtoMesh Generator service, send the IFC data, monitor the conversion process, download the OBJ, MTL and XML files, parse them and import the objects into the currently active scene. The result of the code example with the school project IFC is the following:

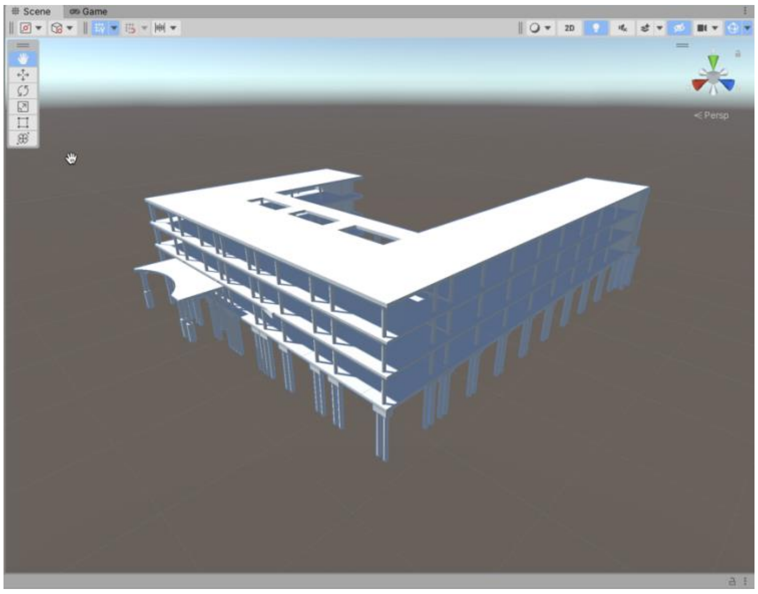


Figure 26 – Loaded object from IFC conversion example code

NOTE: Due to the way that Unity handles translation of objects, the imported object might initially appear rotated by 90 degrees in the X axis. Adding a rotation of -90 in the X axis should reorient the object correctly.

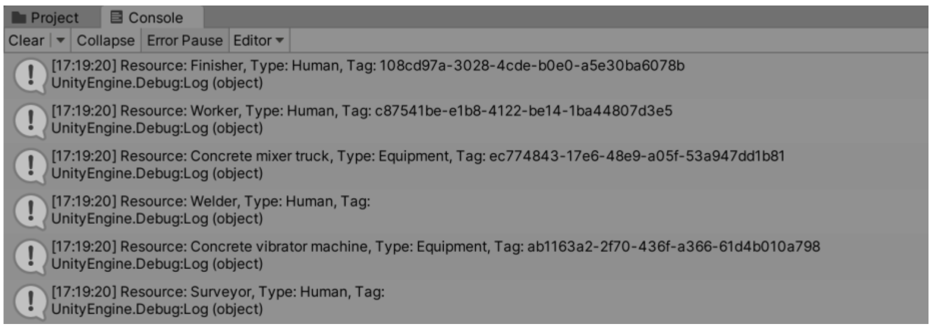
2.5.5 Fetching real-time data

Use the following code to obtain Resource Tracking data.

```
…
var DataManagementComponent =
    GameObject.Find("COGITOGameObject").GetComponent<RealTimeDataManagement>();

var resources = await DataManagementComponent.GetAllResources();
        foreach (var resource in resources)
        {
            Debug.Log($"Resource: {resource.Name}, Type: {resource.Type ==
ResourceType.HUMAN ? "Human" : "Equipment"}, Tag: {resource.Tag}");
        }
```

The above example will output the following (Figure 27):



```
[17:19:20] Resource: Finisher, Type: Human, Tag: 108cd97a-3028-4cde-b0e0-a5e30ba6078b
UnityEngine.Debug:Log (object)
[17:19:20] Resource: Worker, Type: Human, Tag: c87541be-e1b8-4122-be14-1ba44807d3e5
UnityEngine.Debug:Log (object)
[17:19:20] Resource: Concrete mixer truck, Type: Equipment, Tag: ec774843-17e6-48e9-a05f-53a947dd1b81
UnityEngine.Debug:Log (object)
[17:19:20] Resource: Welder, Type: Human, Tag:
UnityEngine.Debug:Log (object)
[17:19:20] Resource: Concrete vibrator machine, Type: Equipment, Tag: ab1163a2-2f70-436f-a366-61d4b010a798
UnityEngine.Debug:Log (object)
[17:19:20] Resource: Surveyor, Type: Human, Tag:
UnityEngine.Debug:Log (object)
```

Figure 27 – Get all resources code example output

2.6 Requirements Coverage

Table 10 presents the results of COGITO Unity Library in conjunction with the DCC requirements coverage analysis. As implied by the requirements coverage status, the second version of the COGITO Unity library is sufficiently developed and might be tested by other Unity application developers within COGITO.

Table 10 – COGITO Unity Library Functional & Non-Functional Requirements coverage from D2.5

ID	Description	Type	Status
Req-1.1	Connect and Authenticate to the DT platform (User login)	Functional	Achieved
Req-1.2	Browse and select project from the list of available projects of the DT platform	Functional	Achieved
Req-1.3	3D visualisation of the infrastructure's geometry (panning, rotation, camera movement and placement, walkthrough) (layer-0)	Functional	Addressed in D7.7
Req-1.4	BIM-elements tree-view and element's selection (layer-0)	Functional	Addressed in D7.7
Req-1.5	Resources tracking data display (layer-1)	Functional	Addressed in D7.7
Req-1.6	QC defects display (layer-2)	Functional	Addressed in D7.7
Req-1.7	H&S issues display (layer-3)	Functional	Addressed in D7.7
Req-1.8	Tasks progress display (layer-4)	Functional	Addressed in D7.7
Req-2.1	Web based App	Non-Functional	Addressed in D7.7
Req-2.2	Scalability	Non-Functional	Achieved, also in tandem with D7.7
Req-2.3	Reusability	Non-Functional	Achieved, also in tandem with D7.7
Req-2.4	Interoperability	Non-Functional	Achieved, also in tandem with D7.7
Req-2.5	Security	Non-Functional	Achieved
Req-2.6	User-friendly	Non-Functional	Addressed in D7.7
Req-2.7	Performance	Non-Functional	Achieved, also in tandem with D7.7

2.7 Development and integration status

As described in the previous sections, in the context of the data transformation for 3D BIM rendering, within COGITO, a Unity package has been developed from scratch; this package aims to serve as a middleware between the DTP and the DCC.

All the modules responsible for the retrieval, loading, transformation, and decomposition of the 3D BIM files and accompanying metadata, schedule information, and the results of the COGITO digital twin applications have been developed and are being tested. Modules facilitating the communication and data exchange with the DTP have been implemented.

Nevertheless, their development progress is tightly coupled with the advances in the development of the DTP and the supporting Digital Twin applications for Health & Safety, Quality Control and Workflow Management. While the data formats of the COGITO components for real-time health & safety and workflow management in the construction phase, as well as their handling by the DTP are progressing, refinements on the corresponding Unity Library's modules might be required.

2.8 Assumptions and Restrictions

The features and implementation described in this deliverable are based on the communication methods and data formats agreed between the DTP and the other relevant tool-developing partners. At the time of writing, preliminary communication tests with the DTP have been performed, whereas the version of the COGITO Unity package that is documented here has driven the development and delivery of the second release of the DCC, described in "D7.8 – Construction Digital Twin 3D Visualisation module v2".

Any modifications or additions imposed by either the DT Platform itself or any of the other COGITO components will be accommodated during next releases of the COGITO Unity Library that will be documented in "D8.2 – Integrated COGITO system v2" and "D8.3 – Integrated COGITO system v3".

3 Conclusions

In this document, the progress made in the context of task T7.3 towards the development of a data transformation module for 3D BIM rendering is reported; in short, the components developed as part of this task are responsible for handling the BIM, schedule and real-time data fetching/extraction, any required transformation and ultimately the preparation of data in a format suitable for visualisation and rendering purposes. The development of the tools and their provided functionality is fully in line with both the stakeholder and the functional/non-functional requirements and ontologies as described in the corresponding deliverables.

After a concise presentation of the conceptual architecture employed throughout the development of the tools, i.e., the COGITO Unity Library and the IFCtoMesh Generator service, their offered functionality as well as their interactions with the DT platform are elaborately presented. The technology stack and all implementation details as well as a usage walkthrough (including a description of the input/output and exposed APIs) of the tools developed are also included in the present deliverable.

Both COGITO Unity Library and DCC are merely data consumers of the DTP data. The pilot sites present individual challenges that may require the DTP and its main data providers (Health & Safety, Quality Control, and Workflow Management components) functionalities adaptation to their specific needs. This requirement is evident from the early stages of the project. The exact needs and specificities are still being recorded and analysed. Consequently, the DTP and other COGITO components might have to be refined later to fully address the pilot demonstration needs. Hence, any COGITO Unity Library's modifications or additions imposed by such refinements will be documented and reported in "D8.2-Integrated COGITO system v2" and "D8.3-Integrated COGITO system v3," planned to be released in M30 (April 2023) and M34 (October 2023), respectively.

COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu