# Intelligent Verification/Validation for XR Based Systems

**Research and Innovation Action**

Grant agreement no.: 856716

# D5.3 – Full integration of the pilots

**iv4XR – WP5 – D5.3**

**Version 1.4**

**February 2022**

| | |
|---|---|
| Project Reference | EU H2020-ICT-2018-3 - 856716 |
| Due Date | 31/09/2021 |
| Actual Date | 10/2/2022 |
| Document Author/s | Joseph Davidson (GA), Fernando Pastor (UPV), Wishnu Prasetya (UU), Jean-Yves Donnart (THA-AVS), Marta Couto (INESC-ID), Tanja Vos (UPV), Jason Lander (GW), Ian Saunter (GW) |
| Version | 1.4 |
| Dissemination level | Public |
| Status | Final |

This project has received funding from the European Union's Horizon 2020 Research and innovation programme under grant agreement No 856716

| Document Version Control | | | |
|---|---|---|---|
| **Version** | **Date** | **Change Made (and if appropriate reason for change)** | **Initials of Commentator(s) or Author(s)** |
| 1.0 | 14/09/2021 | Initial document structure and contents | JD |
| 1.1 | 23/09/2021 | FTAs integration with SE, GA work | JD, FP, WP |
| 1.2 | 24/09/2021 | GA/THA-AVS finished first draft of work | JD, JYD |
| 1.3 | 30/09/2021 | GA/THA finalize report, add more up to date diagrams. | JD, JYD, TV |
| 1.3 | 30/09/2021 | GW Report, Final conclusion written | JL, IS, JD |
| 1.35 | 17/01/2022 | Reopened for changes, Adding GA manual and examples | JD, FP |
| 1.4 | 27/01/2022 | Finalised GA changes to main text, added example for users. | JD |
| 1.4 | 31/01/2022 | GWE/THA-AVS additional materials | JL, IS, JYD |

| Document Quality Control | | | |
|---|---|---|---|
| **Version QA** | **Date** | **Comments (and if appropriate reason for change)** | **Initials of QA Person** |
| 1.2 | 27/09/2021 | Polishing of common and GA parts. | JD |
| 1.2 | 29/09/2021 | Formatting of links, captioning of images | MC, JD |
| 1.2 | 30/09/2021 | Change of some diagrams | TV, WP, JD, JYD |
| 1.3 | 30/09/2021 | Cleanup for final submission | JD |

| | | | |
|---|---|---|---|
| 1.4 | 10/02/2022 | Final version of resubmission | JD |

| Document Authors and Quality Assurance Checks | | |
|---|---|---|
| **Author Initials** | **Name of Author** | **Institution** |
| JD | Joseph Davidson | GA |
| FP | Fernando Pastor | UPV |
| WP | Wishnu Prasetya | UU |
| JYD | Jean-Yves Donnart | THA-AVS |
| MC | Marta Couto | INESC-ID |
| TV | Tanja Vos | UPV |
| JL | Jason Lander | GW |
| IS | Ian Saunter | GW |

**TABLE OF CONTENTS**

**1 Executive Summary** 1
Acronyms and Abbreviations 1

**2 Introduction** 2

**3 Full Integration definition** 3
3.1 Space Engineers (Good AI) 3
3.2 Nuclear plant intrusion simulation (Thales AVS) 4
3.3 LiveSite (Gameware) 5

**4 Space Engineers (Good AI) Integration** 7
4.1 The Example Scenario 7
4.2 Functional Test Agents Integration 7
4.3 Further Work 10
4.3.1 Block Photoshoot 11

**5. Nuclear plant intrusion simulation (Thales AVS) Integration** 12
5.1 Description of the Test Scenario 12
5.2 Full Integration with the Framework 13
5.2.1 Accelerating the simulation 15
5.2.2 Expanding the interface 16
5.3 Further Work 16

**6. LiveSite (Gameware) Integration** 17
6.1 Overview 17
6.2 Reading verification and analysis 17
6.3 Integration Overview 18
6.4 Multi-Sites 22
6.5 Full Integration for Test type 2 - Train moving over structure 23
6.6 Further work 24

**7 Conclusions** 26

**A1 Space Engineers Plugin User Manual** 27
1.1 Introduction 27
1.2. How to run the game with this plugin 28
1.2.1 3rd Party Dependencies 28
1.3 How to build 29
1.4 API 30
1.4.1 Architecture Overview 30
1.4.2 Project details: Ivxr.SePlugin 31
1.5 Space Engineers Engine information 31
1.5.1 Units and position 31

1.5.2 Character and camera orientation 32
1.5.3 Basic movement API 32
1.5.4 Movement types and speed 32
1.5.5 Continuous movement 33
1.5.6 Blocks 33
1.5.6.1 DefinitionId, id 34
1.5.6.2 Block-specific instance properties 34
1.5.6.3 Definitions 35
1.5.6.4 Small vs large cube blocks 35
1.5.6.5 Targeting a block 35
1.5.6.6 Using blocks 36
1.5.6.7 Compound blocks 36
1.5.7 Welding and grinding 36
Shared constants: 37
Angle grinder constants and formula 37
1.5.8 Advanced/plugin customization - Adding new block-specific fields 38
Base interface 38
Polymorphic blocks 38
1.5.9 Multiple characters 38
Commands unaffected by character switch 39
1.6 Plugin Examples 39

**A2 Nuclear plant intrusion simulation User Manual** 48
2.1 Introduction 48
2.2 Tests Procedure 48

# 1 Executive Summary

This report details the progress of the integration of the iv4XR framework with the pilots supplied by the industrial partners. It explains what full integration entails for each of the pilots, presents an overview of the technical details of this integration for each pilot, and finally describes how each pilot will make use of the interfaces going forward and outlines any future support and development this may entail. For the integrations where it makes sense, there are also user manuals for the pilots appended to this document.

### Acronyms and Abbreviations

| | |
|---|---|
| **XR** | Extended Reality |
| **GA** | GoodAI |
| **SE** | Space Engineers |
| **CGE** | Computer Generated Entities |
| **CGF** | Computer Generated Forces |
| **FTA** | Functional Test Agent |
| **RL** | Reinforcement Learning |

## 2 INTRODUCTION

One of the objectives of the iv4XR project is to permit external organisations to use the framework so that their extended reality environments can be tested or monitored with less human interaction than is required by the testing methods of today. For adoption to be effective, prospective developers need 1.) Some demonstration of the benefits of using iv4XR, and 2.) A measure of guidance on how to integrate the framework into their development lifecycle. The pilots are one of the methods that the consortium is using to deliver this knowledge.

There are three integration deliverables in this project, of which this is the third. The work in this deliverable follows the previous deliverables in bringing the features of the interfaces into line with the general capabilities of the iv4XR framework.

With this deliverable, the features required for the interfaces to the pilot are complete so that in theory, all aspects which we would wish to test in the pilots are able to be tested. However, we anticipate that further work will be needed to support more features. This anticipated future work will be addressed for each pilot.

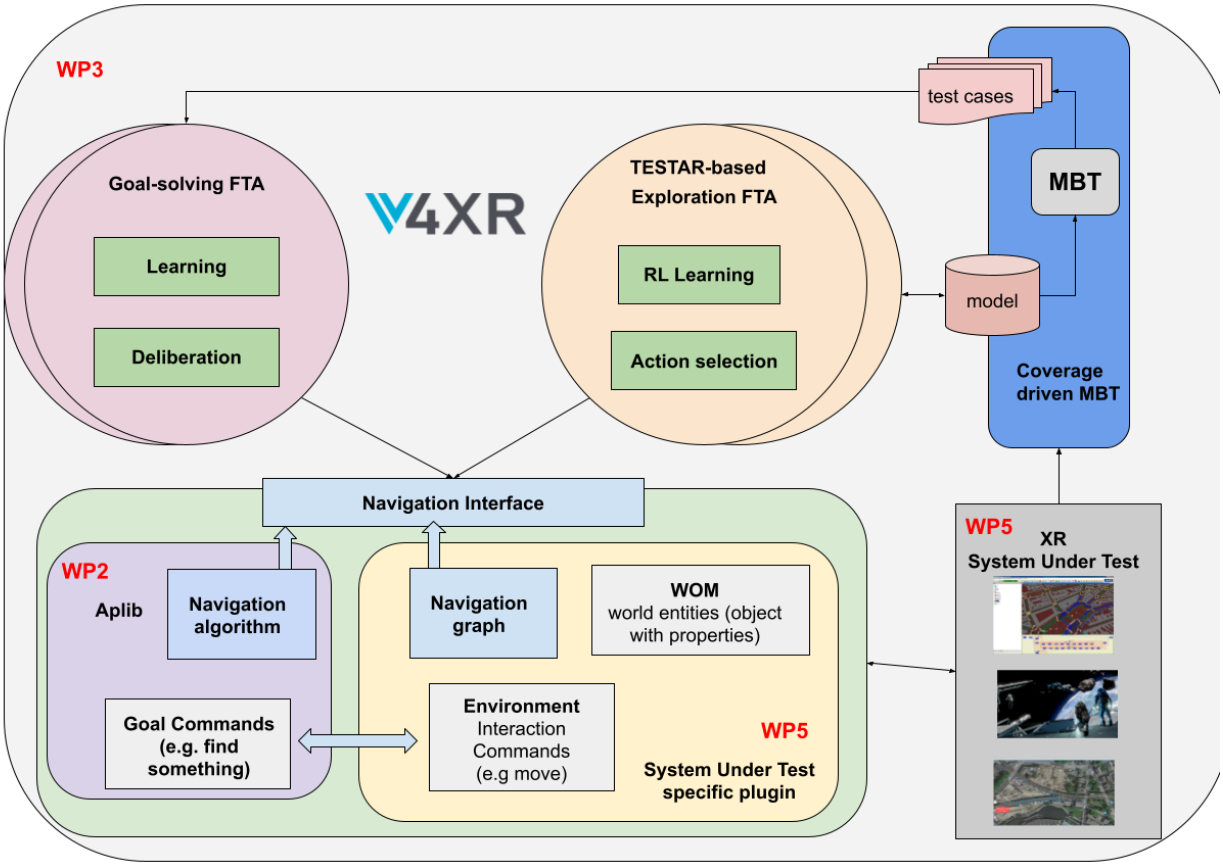



Diagram of how the iv4XR framework interfaces with the pilots.

# 3 Full Integration definition

In previous deliverables, basic integration and intermediate integration have been broadly described as “one-way communication” and “two-way communication” respectively. With two-way communication, the pilots are able to host agents which are able to influence and be influenced by the pilot environment. Full integration concludes by exposing all of the relevant controls that the framework will need in order to control the pilots, read data, and verify correctness.

In the previous deliverable, basic integration was broadly expressed as “one way communication” between the framework and pilot where the pilot sends some observation to the framework via the interface. Intermediate integration was similarly defined as “two way communication” between the framework and pilots.

For full integration, we are concentrating on a “feature complete” version of the interfaces so that the developer of a test agent has access to all of the functionality and internal information required in order to test the salient features of the pilot.

## 3.1 Space Engineers (Good AI)

As a game under continued development, Space Engineers requires thorough and frequent testing. Given the complexity of the game world, the testing team performs the majority of the tests manually. One of these tests is in the creation of game entities known as “blocks”. Blocks are the basic construction components from which the players can build structures and vehicles. Blocks vary in functionality from generic armour blocks to more complex storage and medical unit blocks.

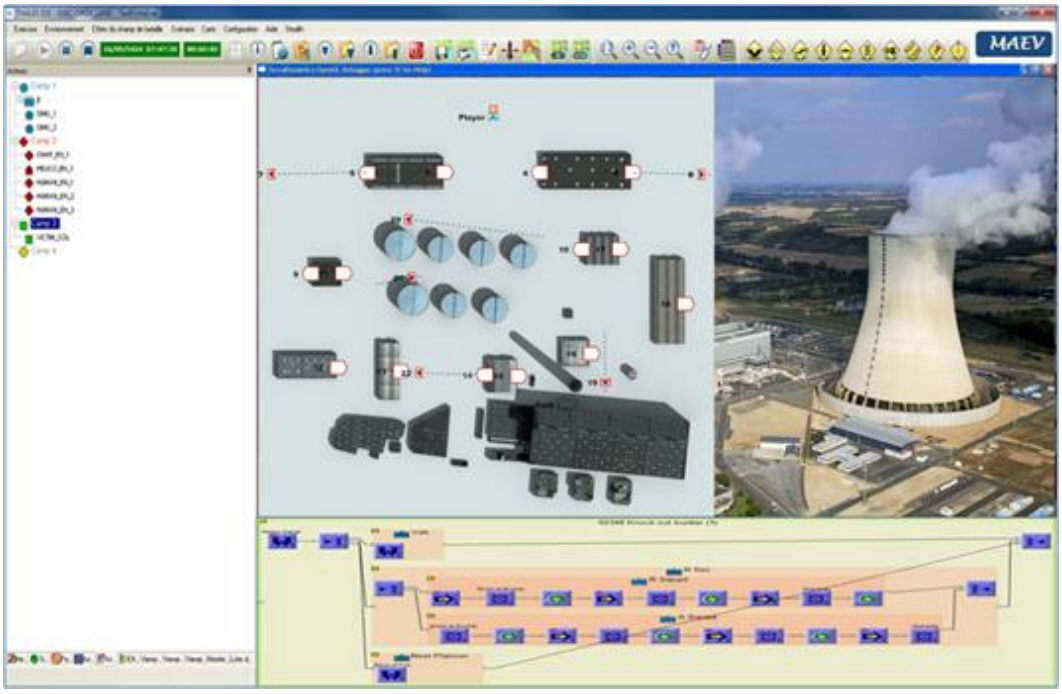

A collection of different blocks; A gyroscope (left), a reactor (centre), and a thruster (right) all attached to a platform of armour blocks.

In previous deliverables, we have focused on a specific test case for determining if a block can be constructed using the current materials in the inventory. For this deliverable, we have expanded the functionality of the interface so that agent developers are able to access mutable and static properties of any block in the game. We have also extended the welding/grinding usecase towards verifying the textures used.

The academic consortium partners have been making use of the interface for their experiments and have been constructing agents to interact with SE. This has been a way to generate feedback for the interface and for ideas to improve the workflows of those that create agents.

### 3.2 NUCLEAR PLANT INTRUSION SIMULATION (THALES AVS)

Today, the verification of a simulation with artificial agents (non-player characters) often relies on human operators which play the opposite force (e.g., the intruders) with the objective to challenge the scenario integrated into the simulation. Because this task is cost consuming, we sometimes use the scenario functionalities of our CGE MAEV to model the different strategies of the opposite force and analyze, afterwards, the results of the different confrontations. In most cases, this kind of verification is not exhaustive enough to be efficient.
Our main objective is iv4XR Project is to test an alternative solution for scenario verification where AI agents can learn how to challenge the CGE scenario simulating the defense of a nuclear plant. These AI agents will have access to all the necessary information (the perceived situation of each intruder) and will be allowed to decide which actions will be performed by their corresponding artificial agents into the simulation.




Use of CGE MAEV to simulate a Nuclear Plant Intrusion

During the “Basic Integration” phase, we defined an API plugin (AIEngine) that allows our CGE MAEV to communicate with AI agents, whatever AI technology they rely on, and developed the communication modules both in the CGE and in the IV4XR Framework in order to ensure the genericity of the communication.

During the “Intermediate Integration” phase, we have tested the capacity for an external module, developed by another partner of the consortium (Thales SIX), to control some of the MAEV agents through the IV4XR Framework. Concretely, this means that this external module is able to receive the states and detections of the MAEV agents under its control and is also able to give high level commands, such as “MoveTo”, to these agents.

Our objective, during the “Full Integration” phase, was to fulfill all the requirements needed to allow an external AI tool, such as Thales SIX Reinforcement Learning (RL) algorithms, to challenge the defense strategy implemented in MAEV.
To achieve this objective, the CGE should be able to run the simulation much quicker than real time in order for the RL algorithms to test and evaluate thousands of alternatives as quickly as possible.
We have also expanded the capacities of the interface in order for the AI tools, not only to control MAEV agents, but also to control the course of the simulation and to access the simulation data that are needed to evaluate the alternatives.

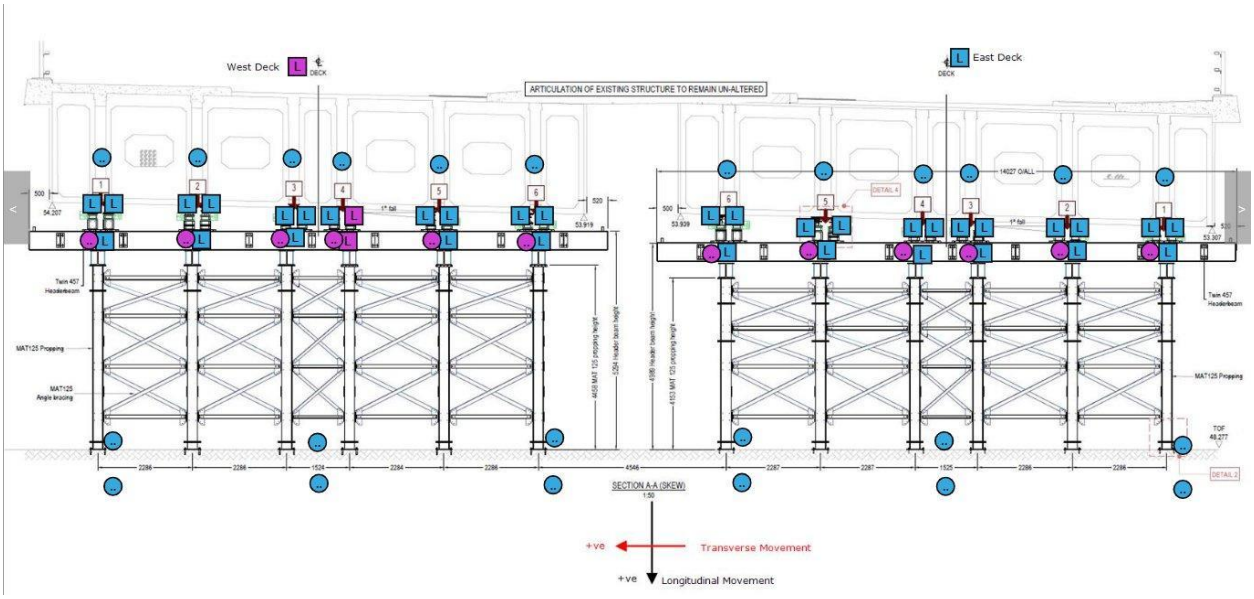
In further works, we will continue to support our partners in order to allow them to perform their experiments using our pilot, and we will work on collective demos to exhibit the results of these experiments.

### 3.3 LIVESITE (GAMEWARE)

For the basic integration phase, we developed a server-side tool which can interface with the IV4XR framework. Its inputs are monitoring projects with sensor definitions, thresholds and their varying requirements, and it uses the IV4XR framework to test parameters within the definition of the given sensors.

The server-side tool is written in JAVA and processes command files which include zero or more data items from the project itself. Since the monitoring projects from which we sample frequently have large amounts of data (often gigabytes), the LiveSite tool processes only a small amount of it at any one time.

For the intermediate integration phase, our objective has been to further enhance this tool to allow both processing and navigation of the project, by allowing the tool to control which sections of the data it is looking at. In effect, it is an agent navigating through the data.
Also for some engineering projects, for example where there is monitoring of a train going over a bridge or section of track, there is a physical moving entity as well as the structure itself. For this phase we have been adding support to enable monitoring of moving entities in the project.




An overview of a rail bridge and the locations of various sensors that monitor the bridge.

For full integration we have advanced the system to analyse the formulae for inter-dependent sensors which are frequently found on large structures such as bridges and buildings. These typically use algebraic relationships such as DISPLACEMENT_PIER1 = tan(ANGLE_PIER1)*LVDT_PIER1

The JAVA tool now creates agents which iterate over the formulae, checking all component sensors are working and verifying the resulting sensor readings by calculating the output itself as well as looking at the output values stored.

We have also split the system into multiple layers, with the top layer controlling the overall checking of a site, which involves issuing commands to perform further checks on various sensors and time-slices. Each of the further checks uses a different agent with a small set of test criteria to validate. The overall results from the checks are then combined once complete, allowing the system to make maximum concurrent use of the server bandwidth available.

# 4 Space Engineers (Good AI) Integration

The majority of the effort that has been expended on the interface in this deliverable has been to do with facilitating how the character of the agent can interact with the world, and how the agent can read data from the game environment. Furthermore, we have added additional helpful functions which are intended to ease the writing of new agents. This has enabled us to extend the welding/grinding case so that screenshots can be taken of the intermediate textures, and these screenshotted textures checked to see if they are the correct ones. This “block photoshoot” is currently being developed in collaboration with another EU project.

## 4.1 The Example Scenario

One of the example scenarios that has been developed for testing the plugin in a manner similar to the current manual testing is the “Place, Grind, and Weld” scenario. In this scenario the agent places a block in creative mode, equips the grinder, deconstructs the block until the integrity of the block is at a certain level, equips the welder, and uses the welder to repair the block to full integrity.

In this scenario, the agent is primed with goals and tactics which align with the tasks given above and these tactics continuously monitor the attribute of interest (in this case the integrity of the block) and evaluate the tests at the appropriate points in the execution.

## 4.2 Functional Test Agents Integration

There are two types of functional test agents (FTAs): goal solving and exploratory FTAs. A goal-solving agent performs tests according to the specified goals. An exploratory agent on the other hand tries to explore and interact with the target system as much as possible, e.g. to check that a property remains robust under unexpected interactions. These FTAs are integrated with the SE System Under Test (SUT) as follows:

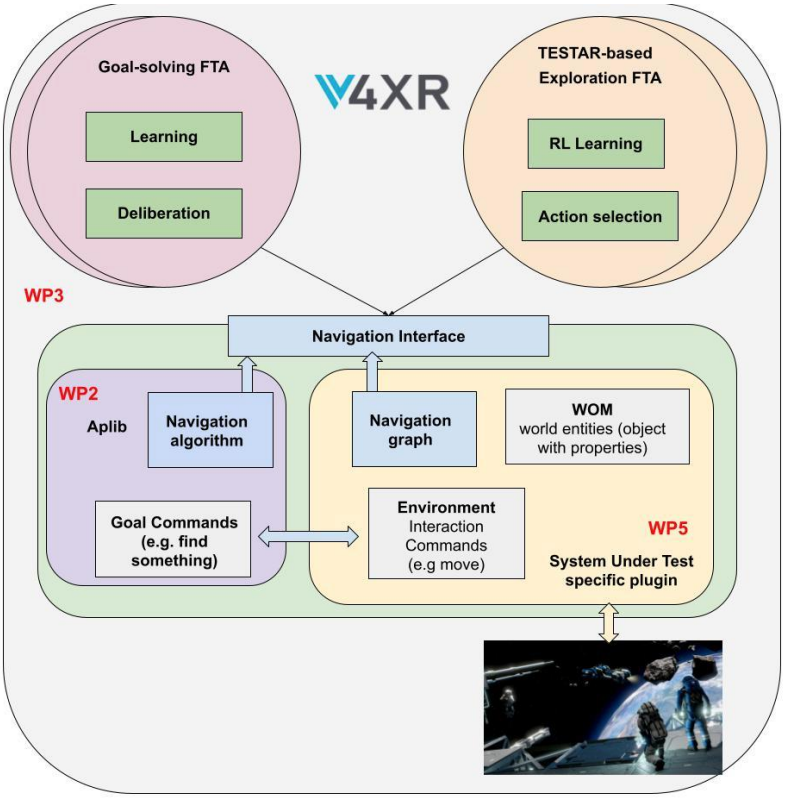



Diagram of how the iv4XR framework interfaces with SE.

Inside the **SUT-specific WOM component** there is the following interface to obtain properties about the SE entities:

- The `Observer` interface allows FTAs to obtain information about the existing entities (types of blocks, other characters) as well as the properties of the blocks and the agent itself (block integrity, agent health, position, orientation, etc...) in a customizable observation ratio.

Inside the **SUT-specific Environment**, there are implemented interfaces for the following interactions:

- The `Character` interface allows FTAs to rotate to and orient themselves, move and fly in the three-dimensional space of the game, as well as interact and use tools on the different types of observed blocks.
- The `Items` interface allows FTAs to select any type of tool for its usage, or existing block type for its placement.
- The `Blocks` interface allows FTAs to place blocks.
- There is an `Admin` interface which allows FTAs to modify the game state without going through the actions nominally available to players. Examples of this are spawning blocks, and teleporting characters or grids.

Related to **Navigation,** the UU is currently integrating an on-line sparse 3D grid algorithm, from which 2D as well as 3D pathfinding (with jetpack) can be performed. The algorithm does not require a pre-constructed navigation mesh, but instead constructs a sparse grid on-the-fly as the agent moves around and avoids obstacles. This will allow FTAs agents to have better movement navigation around the environment to solve test goals or to execute more intelligent exploratory movements.

To see the goal-solving FTA in action we show below an example of a test-goal that can be given to the agent. The goal will guide the agent to auto-navigate to an in-game battery unit, and then grind it. It checks two correctness assertions about the integrity of this battery unit, and also takes screenshots for inspection by testers. A video[1] shows the run.

```
GoalStructure G =
SEQ(
  DEPLOYonce(agent, closeTo(agent, "LargeBatteryBlock" ...)),
  targetBlockOK(agent, e →
    (float) e.getProperty("integrity") == (float) e.getProperty("maxIntegrity")),
  photo("LargeBattery.png"),
  grinded(agent,0.5),
  targetBlockOK(agent, e →
    (float) e.getProperty("integrity") ≤ 0.5 * (float) e.getProperty("maxIntegrity")),
  photo("GrindedLargeBatteryGrinded.png"))
```

A listing of the algorithm used by the agent that grinds and screenshots a battery block.

To see the exploration FTA in action we can see in this video[2] how the TESTAR Spy mode uses the iv4XR SE plugin to fetch the State of the observed blocks, and how the Generate mode allows TESTAR to move to the target blocks dealing with obstacles to finally execute an interaction like the grid of the block.

For the “Place, Grind, and Weld” scenario the goals and tactics can be expressed in Kotlin thus:

```
val testingTask: GoalStructure =  SEQ(
        goals.agentAtPosition(Vec3(532.7066f, -45.193184f, -24.395466f), epsilon = 0.05f),
        goals.agentDistanceFromPosition(
            Vec3(532.7066f, -45.193184f, -23.946253f),
            distance = 16f,
            epsilon = 0.1f,
            tactic = tactics.moveForward(),
        ),
        goals.blockOfTypeExists(
            blockType,
            tactic = tactics.buildBlock(blockType),
        ),
        goals.lastBuiltBlockIntegrityIsBelow(
            percentage = 0.1,
```

[1] https://www.youtube.com/watch?v=d9IBT1TfecU
[2] https://www.youtube.com/watch?v=HKWsjWV0hmo

```
            tactic = SEQ(
                tactics.equip(grinderLocation),
                tactics.sleep(500),
                tactics.startUsingTool(),
            ),
        ),
        goals.alwaysSolved(
            tactic = SEQ(
                tactics.endUsingTool(),
                tactics.sleep(500),
            ),
        ),
        goals.lastBuiltBlockIntegrityIsAbove(
            percentage = 1.0,
            tactic = SEQ(
                tactics.equip(welderLocation),
                tactics.sleep(500),
                tactics.startUsingTool(),
            ),
        ),
        goals.alwaysSolved(
            tactic = SEQ(
                tactics.endUsingTool(),
            ),
        ),
    )

    testAgent.setGoal(testingTask)
```

Here the agent is primed to construct a particular type of block, grind the block down and weld it back up to 100% integrity in sequence. Once a goal is achieved, the agent stops with its current tactic.

### 4.3 Further Work

The SE interface is being actively worked on to include other features that are not essential to the testing of SE, but will be helpful for functional or social agent developers.

Earlier in this deliverable, the partners at UU described the development of a navigation system for SE based within the framework. One of the first improvements to the interface is planned to be a navigation graph for grids of blocks which can then be used for planning movement around test cases.

The work into SETAs is starting to involve SE. The interface as it currently stands reports both the internal state and status of the agent in relation to the environment. One thing that is not really reflected in any of the API's is the user interface as everything is accessed programmatically. Future work will extend the interface to represent the UI and its events.

The plugin itself is currently undergoing trials with the game testing team to see if it is convenient for the testers to automate some of their easier repetitive tests. Current work in this space is in rewriting their human-centric testing scenarios to something that is easier to automate, and in developing multi-agent control so that the simulation integrity can be verified

when testing. The current results are promising and the next deliverable (D5.4: Project validation report) will contain a report arising from these efforts.

### 4.3.1 Block Photoshoot

GA participates in another H2020 Project: VeriDREAM[3], and collaboration here has exposed some additional interesting test cases in SE for which iv4XR can be used. In this test case, the testers verify that the texture progression from the initial stages of construction to the finished product have been rendered correctly.

To automate this, we have an agent which can spawn a block fully built, then it uses a grinder on the block to deconstruct it, where at each stage it moves back and takes a picture of the block. There were several iterations of this behaviour. The first involved cube blocks that were placed in a room where an api call changes the background blocks to a pink colour so that the actual texture of interest could be identified easily.

The block of interest highlighted apart from the coloured background.

While this worked for cube blocks, there are larger blocks where it was impractical to back up a room in order to get the whole texture in frame without finetuning the amount that the character has to back up by. In this case the testing moved to space, where the agent is moving via its backpack and the black backdrop of space providing a contrast so that the textures can be identified easily. A video of this iteration can be seen here[4].

The third and current iteration of the photoshoot involved the implementation of the teleportation api, where the agent developer can specify position and orientation vectors for the character, and the game will place them there. This was done so that blocks could be imaged from all sides, rather than just one. A video of this iteration can be seen here[5].

The block photoshoot is still in development. We are waiting for the texture comparison method and have some improvements planned such as having flat ambient light to illuminate all sides of a block at once for clear imaging of the textures.

[3] VeriDREAM project (grant agreement no. 951992) https://www.veridream.eu/
[4] https://www.youtube.com/watch?v=wNmi3OOZcaU
[5] https://www.youtube.com/watch?v=Vz5Fm40ZMiM

# 5. NUCLEAR PLANT INTRUSION SIMULATION (THALES AVS) INTEGRATION

In this pilot, the testing agents are AI agents that try to defeat the defense strategy of a power plant by finding a way to get an intruder to the core of the plant without being detected. Because the number of guards and cameras are limited, our objective is to use AI tools to test the quality of different defense strategies in order for the operator to find a compromise between efficiency and available resources.

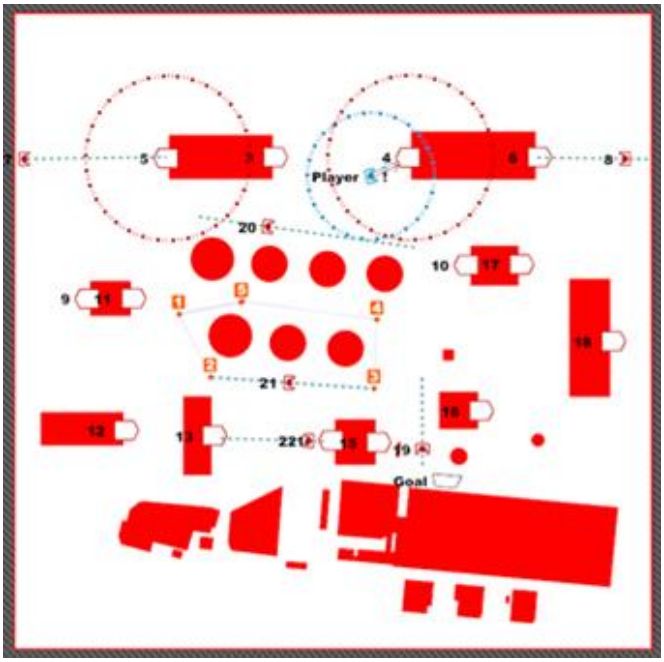
## 5.1 DESCRIPTION OF THE TEST SCENARIO

Within this scenario, intruders may use a map of the plant but have no initial information about the locations of cameras and about the number and behavior of the guards.

The defense strategy is modeled by Thales' simulation tool MAEV thanks to its scenario module where one can position cameras and assign missions to the guards (such as patrol). The behavior of the guards is modeled in MAEV thanks to traditional AI capacities like mission graphs, behavioral trees and pathfinding algorithms.

The following figure represents a map of the plant, and we can see a basic scenario that has been modeled during the "Intermediate Integration" phase:

- Agent 1 (the "Player") represents an intruder who tries to reach the "Goal". It is the only MAEV agent controllable by an external module.
- Agents 3, 4, 5, 6, 9, 10, 11, 12, 13, 15, 16, 17, 18 and 19 are 180° cameras;
- Agents 7, 8, 19, 20, 21 and 22 are guards with specific missions (e.g., agent 21 makes a circle patrol passing by locations identified by squares 1, 2, 3, 4 and 5 on the map).




A depiction of the environment in which the agents operate.

Each agent has a detection range, symbolized by a circle on the map, that parametrizes the detection capacity of the agents. For instance, we see in the figure that the “Player” is detected by Camera 4 but the camera is detected in return by this agent.

## 5.2 Full Integration with the Framework

The pilot offers the possibility for an external module, for instance AI agents, to control some of the MAEV agents through the IV4XR Framework. The following figure describes the integration of the Framework with both MAEV and the AI Tools that is currently used by THALES-SIX to perform the tests.

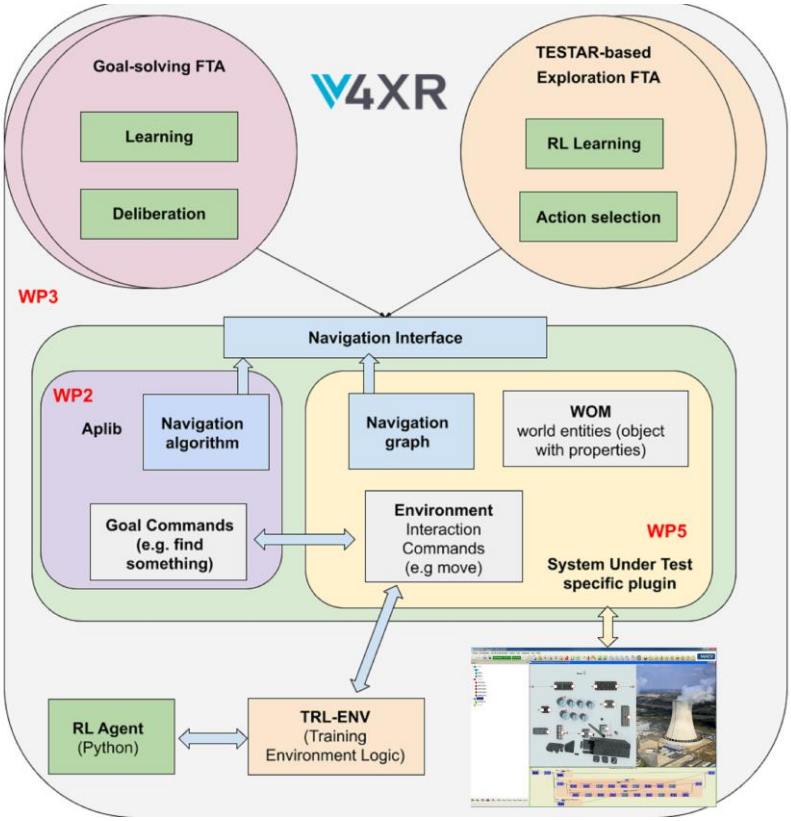



Diagram of how the iv4XR framework interfaces with MAEV and Thales-SIX AI tools

This communication with the Framework IV4XR is performed by TCP sockets and use the following JSON format:

- For sending commands from the Framework to MAEV agents:

{"cmd":"AGENTCOMMAND","arg":{"cmd":"MOVETO","agentId":1,"targetId":1,"arg":{"x":1.0,"y":0.0,"z":0.0}}
}
{"cmd":"AGENTCOMMAND","arg":{"cmd":"DONOTHING","agentId":1,"targetId":1,"arg":{"t":1.0}}}

- For receiving MAEV agent's state vectors by the Framework:


```
{"agentID":1,"tick":13139012,"agentPosition":{"x":0;,"y":0;,"z":0;},"velocity":{"x":10;,"y":10;,"z":0;},
  "didNothing":"true",
  "entities":[
        {"isActive":"true",”center":{"x":5;,"y":0;,"z":5;},"extents":{"x":0,"y":0.,"z":0.},"type":2,"tag":"","id":
        2,"position":{"x":5;,"y":0;,"z":5;},"property":""},
        {"isActive":"true","center":{"x":3;,"y":0;,"z":2;},"extents":{"x":0.,"y":0.,"z":0.},"type":2,"tag":"","id":
        3,"position":{"x":3;,"y":0;,"z":2;},"property":""},
        {"isActive":"true",”center":{"x":4;,"y":0;,"z":2;},"extents":{"x":0.,"y":0.,"z":0.},"type":2,"tag":"","id":
        4,"position":{"x":4;,"y":0;,"z":2;},"property":""}],
"navMeshIndices":[]}
```


During the “ Intermediate Integration " phase, a first version of the pilot, integrating MAEV with the above scenario to the IV4XR Framework, was embedded on a PC and sent to Thales SIX for basic experimentations.

The results are in line with our expectations:

- The intruder (Agent 1) moves on the map according to commands sent by the Thales SIX module connected to the Framework;
- The state vector of Agent 1 received by the Thales SIX module evolves according to its current position, speed and detections in MAEV simulation;
- The cameras still detect the intruder controlled by the Thales SIX module when it enters their detection perimeter.

A video of this agent performing tests can be found here[6].

In this experiment, the agent receives its current perceptions and state from MAEV Simulation through the IV4XR Framework and launches MOVETO commands to random destinations to drive the "Player" entity. Note that the agent's destination changes every 10 seconds to make it more demonstrative.

The code of the Thales-AVS plugin[7] is available on the project’s GitHub page. The code of Thales SIX tools is divided in two modules, available on the project’s Github page:

- The **iv4xr-rl-env**[8] module allows for the definition of a RL environment interface over the iv4XR System Under Test. It also contains a connector to some Python RL Agent training code that provides actions and receives states and rewards.
- The **iv4xr-rl-trainer**[9] module allows the training of Deep RL agents written in the Python language on RL environments run by the **iv4xr-rl-env** module within the iv4XR framework.

[6] https://youtu.be/QPpnRloAr7o
[7] https://github.com/iv4xr-project/iv4XR-IntrusionSimulation
[8] https://github.com/iv4xr-project/iv4xr-rl-env
[9] https://github.com/iv4xr-project/iv4xr-rl-trainer

For the intermediate integration, a placeholder that outputs random actions replaces the actual training code.

For the “Final Integration” phase, we have continued to improve the pilot on two aspects:

- By accelerating the simulation in order to allow the use of Reinforcement Learning;
- By expanding the capacities of the interface in order to allow the AI tools to access more simulation data or parameters

### 5.2.1 ACCELERATING THE SIMULATION

The exploitation of Machine Learning algorithms, such as the Reinforcement Learning (RL) tools used by Thales-SIX, necessitates a very large number of experiments in simulation to converge toward an optimal solution to the studied problem. Here, we want an agent, representing an intruder, to find a way to defeat the current defense strategy implemented in the MAEV scenario.

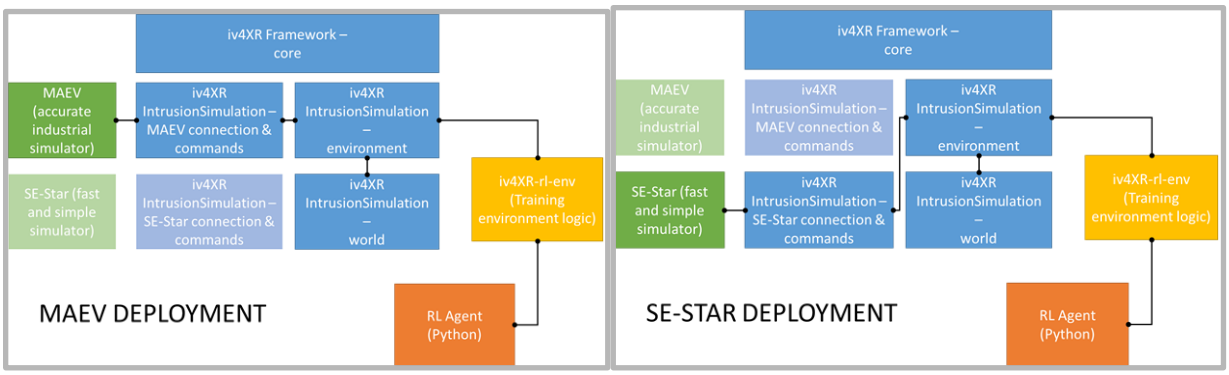
Moreover, after the presentation of several incrementally improved strategies, if the RL tools do not find a way to defeat the defense strategy, we want to be sure that it is because this strategy can not be beaten with only one intruder.

In order to perform all these tests, including this last coverage test, the CGE shall run thousands of iterations of the scenario in order for the RL algorithms to be able to test and evaluate the maximum of alternatives possible.

One solution to this problem is to accelerate the simulation. MAEV simulation has been modified during the “Final integration phase” to run quicker than real time in order to support the Reinforcement Learning requirements.

In the last version delivered to Thales-SIX, there is an “New” Acceleration mode that can be activated thanks, in particular, to the disconnection of the distributive simulation standardized interface HLA and to the differentiation between the simulation time and the clock times in all the models managed by the CGE.

This acceleration of the simulation could have been increased by modifying the level of details of the models in MAEV. Because we want to keep a good level of representativity, we did not want to modify the level of detail of the models. Instead, we have decided to use a simpler simulator (SE-STAR) to perform the early stages of the learning.




An overview of how MAEV and SE-Star interface with the iv4XR framework.

The simulator will also be connected to IV4XR Framework using very similar interfaces to the ones used by MAEV. And after this primary learning stage, MAEV will replace SE-STAR to conclude the learning.

### 5.2.2 Expanding the interface

In order to facilitate the learning process, The AI tools must control not only some of the MAEV agents, but also the simulation itself. The control parameters of the simulation are now accessible through the Framework:

- Load exercise,
- Play & pause,
- Stop & reset,
- Change simulation speed.

The AI tools must also access to more parameters than the detection of the controlled agent:

- The initials positions, field of view and range of all the agents (intruder + guards) and of the cameras;
- The current position and velocity of the guards during the run;
- The current detections made by the guards and the camera in order to calculate the rewards in function of the position of the intruder and the configuration of the environment.
- The current simulation time when simulation speed differs from the real time.

Finally, the final integration ensures the coherence between the representation of the environment in MAEV and the corresponding representation managed by the AI tools (coordinate referential, shape and position of the infrastructures, etc.).

## 5.3 Further Work

We will continue to support our partners during the tests by expanding the interface if necessary and enhancing the level of the Defense Strategy until the RL tools are not able to discover a policy to defeat it.

For the coverage of the tests, we intend to parallelize both simulators to verify that the Defense strategy simulated in MAEV is really the best we can propose for the real infrastructure.

# 6. LiveSite (Gameware) Integration

## 6.1 Overview

Livesite is a real-time instrumentation and monitoring system for the building industry.

Sensors are installed on building and construction sites to monitor ground movements, building movements, temperature variations, rainfall, water levels, vibrations, dust and other environmental factors to ensure safety of both the site being worked on, and the neighbouring area and structures.

The various sensors connect to network hubs and the internet directly and upload readings to the Livesite servers. Readings can be uploaded hundreds of times per second in some instances, and a very large volume of data is produced.

The Livesite servers analyse and process the data, providing real-time graphs, numerical tables, and 3D visualisations of the sensor readings, allowing engineers to monitor the site activity and behaviours.

Ensuring the integrity and accuracy of the readings is critical to provide a safe building environment and prevent works or ground issues from causing problems with the structures involved.

## 6.2 Reading verification and analysis

For IVRX4 we have setup an additional server (the API Server) which can access readings from certain Livesite projects and provides a web-API to access and parse the readings and their associated meta-information.

The API server controls and provides an API to the JAVA tool, which runs tests and queries on Livesite project data and provides results which list errors detected in the project configuration, sensor configurations, or the sensor readings themselves.

The meta-information accompanying a project includes a wide variety of parameters that can be used to assist in verification of the validity of the readings. For instance, the meta for a sensor will include what type of sensor it is, and where it is installed. It can then be deduced that a temperature sensor on the outside of a building for example, should never give a temperature reading significantly higher than the ambient temperature of the area on that particular day.

### 6.3 INTEGRATION OVERVIEW

Using the API server, a script for a Livesite project is provided. This script contains items to test and validate in the given project.

The API Server can provide the readings from sensors with transformations and filtering applied, allowing the IV4XR framework to detect inconsistencies or problems when tested with the JAVA tool command scripts.

The API server can also be used to retrieve tables of readings with meta-information from Livesite projects which can then be analysed by the IV4 framework.

The meta and readings can be downloaded in JSON, XML and raw formats.

For basic integration, the IV4XR framework detected some obvious errors automatically, for example values out of range or missing readings.

Monitoring projects typically run for months or years. The sensor monitoring server executes maintenance tasks over defined periods during the lifecycle of the project, including backups, updating reports, generating alarms etc.

The top layer has been designed to be an additional task which can run alongside the other maintenance tasks, providing a near real-time verification system.

The top layer generates tasks which create agents. Each agent has a set of specific goals, such as verifying a particular sensor's output, or the equation of a sensor, or the settings for a group of connected sensors.

This gives an advantage in that we can control the relative complexity of tasks, and not worry about overloading the system with database requests if such tests were run during periods of intense activity (which can occur in test type2)

Each task is executed when the server iterates over the projects in its management cycle, which is typically between every minute or every few minutes depending on the project requirements.

The task is given a script and access to the relevant project settings/data, so the script can execute entirely independently of any other scripts, allowing easier parallel processing.

The script generates the agents which execute the tests for the specified goal, and then generates output.

Fig. High level script example to test a project

```
PROJECT CANNING_TOWN
[SENSORS]
TEST CONFIGS
TEST EQUATIONS
```

[READINGS]
TEST THRESHOLDS
TEST SANITY
TEST JITTER
TEST DEVICES
TEST EQUATIONS
[DIAGNOSTICS]
TEST ERRORS

Testing equations for the sensors, as part of the full integration, requires parsing and sorting the sensors iteratively and combining test results in a stack. This is due to the fast that sensor equations depend on other sensors, and there maybe various inter-dependences.
The system resolves these until either all are determined valid or omissions remain.

Sensor equations that are invalid may include references to sensors which do not exist, references to themselves, or invalid trigonometric terms.

Simple sensor equation example:

LVDT_PIER_5=LVDT1+LDVT2

This sensor equation shows that the overall displacement of Pier 5 is the sum of the sensors LVDT1 and LDVT2. If either of the missing from the project the equation is invalid.

More complex sensor equation example:

LVDT_PIER_5=SIN(ROT1)*LDVT2

This sensor equation shows that the overall displacement of Pier 5 uses the sine of the value of ROT1 sensor multiplied by LDVT2

Example of error output

PROJECT CANNING_TOWN
ERROR LVDT1 EQUATION MISSING SENSOR LVDT3
ERROR LDVT1 CALIBRATION MISSING
CHECK LDVT1 THRESHOLDS

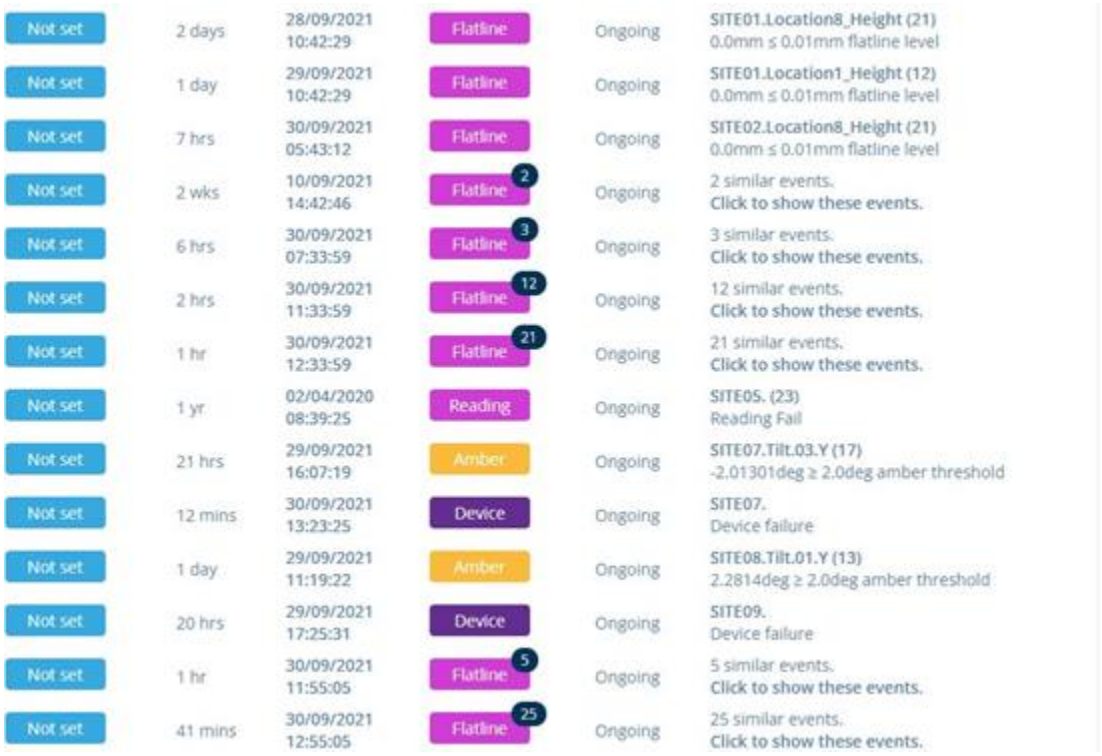
The output contains errors found, as well as requests for further tests. This allows concurrent testing to be done in a hierarchical manner, and also gives the advantage that the system can identify a possible failure even before all subsequent tests are completed.

The device error shown below indicates a problem is occuring on a particular device, more detail will be available after further tests have been run in subsequent tasks, but the engineers know that readings from the device may be invalid and should be ignored until this is verified.

| Status | Started | Date | Type | Duration | Description |
|---|---|---|---|---|---|
| Not set | 1 day | 29/09/2021 08:21:59 | Device | Ongoing | UNIT 1. Device failure |
| Not set | 7 mins | 30/09/2021 13:16:31 | Red | Ongoing | UNIT 2.LAeq (10 Minute) [dB] (16) 92.18dB ≥ 80.0dB red threshold |
| Not set | 7 mins | 30/09/2021 13:16:31 | Amber | Ongoing | UNIT 2.LAeq (10 Minute) [dB] (16) 92.18dB ≥ 78.0dB amber threshold |
| Not set | 30 wks | 26/02/2021 08:47:50 | Reading | Ongoing | UNIT 3.SYSTEM.VOLTAGE (0) Reading Fail |

The hierarchical task and goal approach also allows testing to be adjusted in terms of accuracy, as well as grouping of errors.

Multiple flat-lines over a period of hours or weeks, as shown below, indicate some kind of general problem with communication with the sensor, and it's clearer to see an overview like this, than a long list of individual errors.



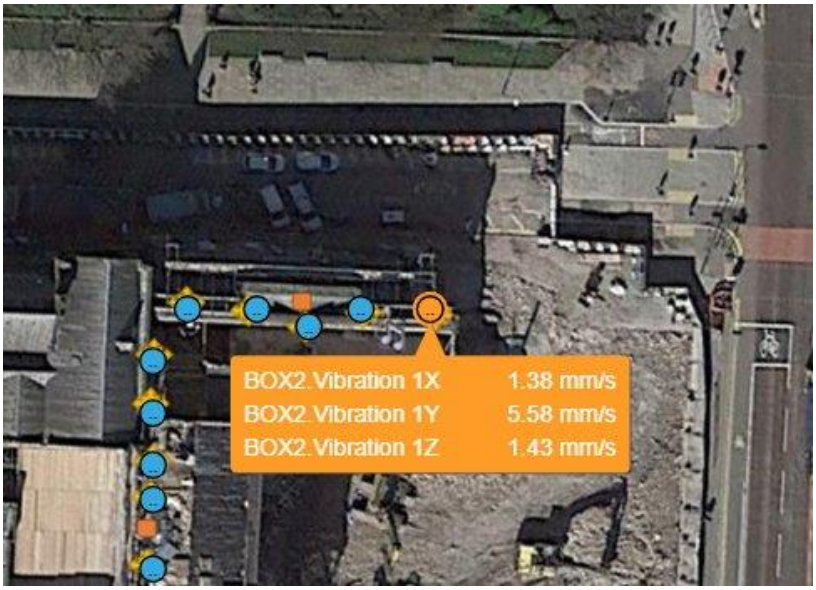
| | | | | | |
|---|---|---|---|---|---|
| Not set | 2 days | 28/09/2021 10:42:29 | Flatline | Ongoing | SITE01.Location8_Height (21) 0.0mm ≤ 0.01mm flatline level |
| Not set | 1 day | 29/09/2021 10:42:29 | Flatline | Ongoing | SITE01.Location1_Height (12) 0.0mm ≤ 0.01mm flatline level |
| Not set | 7 hrs | 30/09/2021 05:43:12 | Flatline | Ongoing | SITE02.Location8_Height (21) 0.0mm ≤ 0.01mm flatline level |
| Not set | 2 wks | 10/09/2021 14:42:46 | Flatline 2 | Ongoing | 2 similar events. Click to show these events. |
| Not set | 6 hrs | 30/09/2021 07:33:59 | Flatline 3 | Ongoing | 3 similar events. Click to show these events. |
| Not set | 2 hrs | 30/09/2021 11:33:59 | Flatline 12 | Ongoing | 12 similar events. Click to show these events. |
| Not set | 1 hr | 30/09/2021 12:33:59 | Flatline 21 | Ongoing | 21 similar events. Click to show these events. |
| Not set | 1 yr | 02/04/2020 08:39:25 | Reading | Ongoing | SITE05. (23) Reading Fail |
| Not set | 21 hrs | 29/09/2021 16:07:19 | Amber | Ongoing | SITE07.Tilt.03.Y (17) -2.01301deg ≥ 2.0deg amber threshold |
| Not set | 12 mins | 30/09/2021 13:23:25 | Device | Ongoing | SITE07. Device failure |
| Not set | 1 day | 29/09/2021 11:19:22 | Amber | Ongoing | SITE08.Tilt.01.Y (13) 2.2814deg ≥ 2.0deg amber threshold |
| Not set | 20 hrs | 29/09/2021 17:25:31 | Device | Ongoing | SITE09. Device failure |
| Not set | 1 hr | 30/09/2021 11:55:05 | Flatline 5 | Ongoing | 5 similar events. Click to show these events. |
| Not set | 41 mins | 30/09/2021 12:55:05 | Flatline 25 | Ongoing | 25 similar events. Click to show these events. |

Testing against years of historical data, with adjusting granularity in time, allows simple overviews of a sensor's state to be determined also, without having to go through all the readings for a particular sensor (of which there could be millions).

Monitoring projects typically save snapshots of readings at various time intervals, such as every minute, as well as every second, or fraction thereof, to minimise database reads when looking back over the data. One may want to see the data for a week overall, in which case looking at readings every 1/100 second is not necessary.

| Status | Started | Date | Type | Duration | Description |
|---|---|---|---|---|---|
| Not set | 5 yrs | 08/07/2016 12:57:47 | Amber 6 | Ongoing | 6 similar events. Click to show these events. |
| Not set | 5 yrs | 08/07/2016 13:27:47 | Amber 3 | Ongoing | 3 similar events. Click to show these events. |
| Not set | 5 yrs | 08/07/2016 14:27:47 | Amber | Ongoing | BOX2_12.1_Long (231) 3.5319mm ≥ 3.0mm amber threshold |
| Not set | 5 yrs | 08/07/2016 14:27:47 | Red 3 | Ongoing | 3 similar events. Click to show these events. |
| Not set | 5 yrs | 08/07/2016 14:57:47 | Red 2 | Ongoing | 2 similar events. Click to show these events. |
| Not set | 5 yrs | 08/07/2016 14:57:47 | Amber | Ongoing | BOX2_11.3_Long (225) 3.43799mm ≥ 3.0mm amber threshold |
| Not set | 5 yrs | 08/07/2016 14:57:47 | Red 2 | Ongoing | 2 similar events. Click to show these events. |
| Not set | 5 yrs | 08/07/2016 14:57:47 | Amber 3 | Ongoing | 3 similar events. Click to show these events. |
| Not set | 5 yrs | 08/07/2016 15:27:47 | Amber 4 | Ongoing | 4 similar events. Click to show these events. |

The hierarchical task testing strategy also allows visualisation of the key issues at any point. At a glance, it is obvious where the problems are on the project as shown below, as the sensor has been marked amber due to excessive vibrations.




The hierarchical testing enables the exact start/endpoint of problems to be identified without requiring full linear search of all the readings in a project, as a tree-based approach is used, with each iteration down reading data with finer granularity.

STRUCTURAL

| Sensor | Alert | Description | Start Time | Repeats | Duration |
|---|---|---|---|---|---|
| LAeq (1 Hour)[dB] | Amber | 85.08dB ≥ 80.0dB amber threshold | 17/09/2021 07:26:47:000 | 2963/1 | Ongoing |

THRESHOLD GRAPH

No readings available.

THRESHOLD TABLE

| Time | Latest | -30 mins | -1 hour | -1.5 hours | -2 hours | -2.5 hours | -3 hours | -3.5 hours | -4 hours | -4.5 hours |
|---|---|---|---|---|---|---|---|---|---|---|
| LAeq (1 Hour)[dB] | No readings | No readings | No readings | No readings | No readings | No readings | No readings | No readings | No readings | No readings |

FUNCTIONALITY

| Sensor | Alert | Description | Start Time | Repeats | Duration |
|---|---|---|---|---|---|
| | Device | Device failure | 17/09/2021 14:30:56:448 | 2536/0 | Ongoing |

SYSTEM HEALTH

No health events.

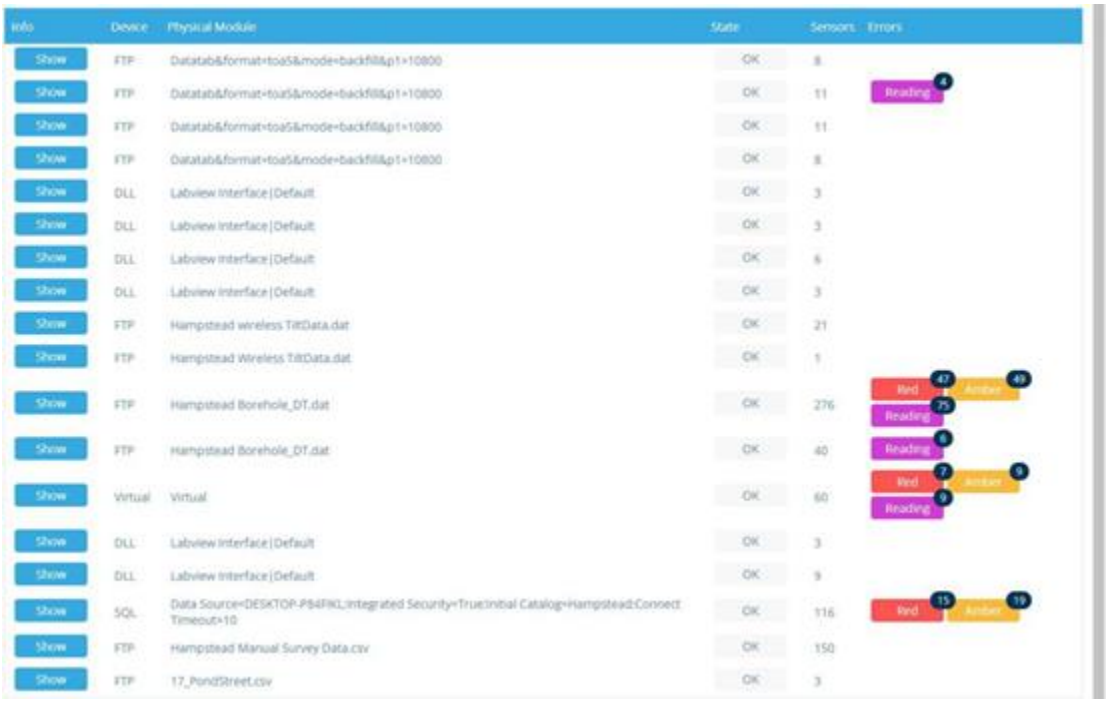
LATEST READINGS AND MIN/MAX/AVG

| Name | Latest | Min | Max | Average |
|---|---|---|---|---|

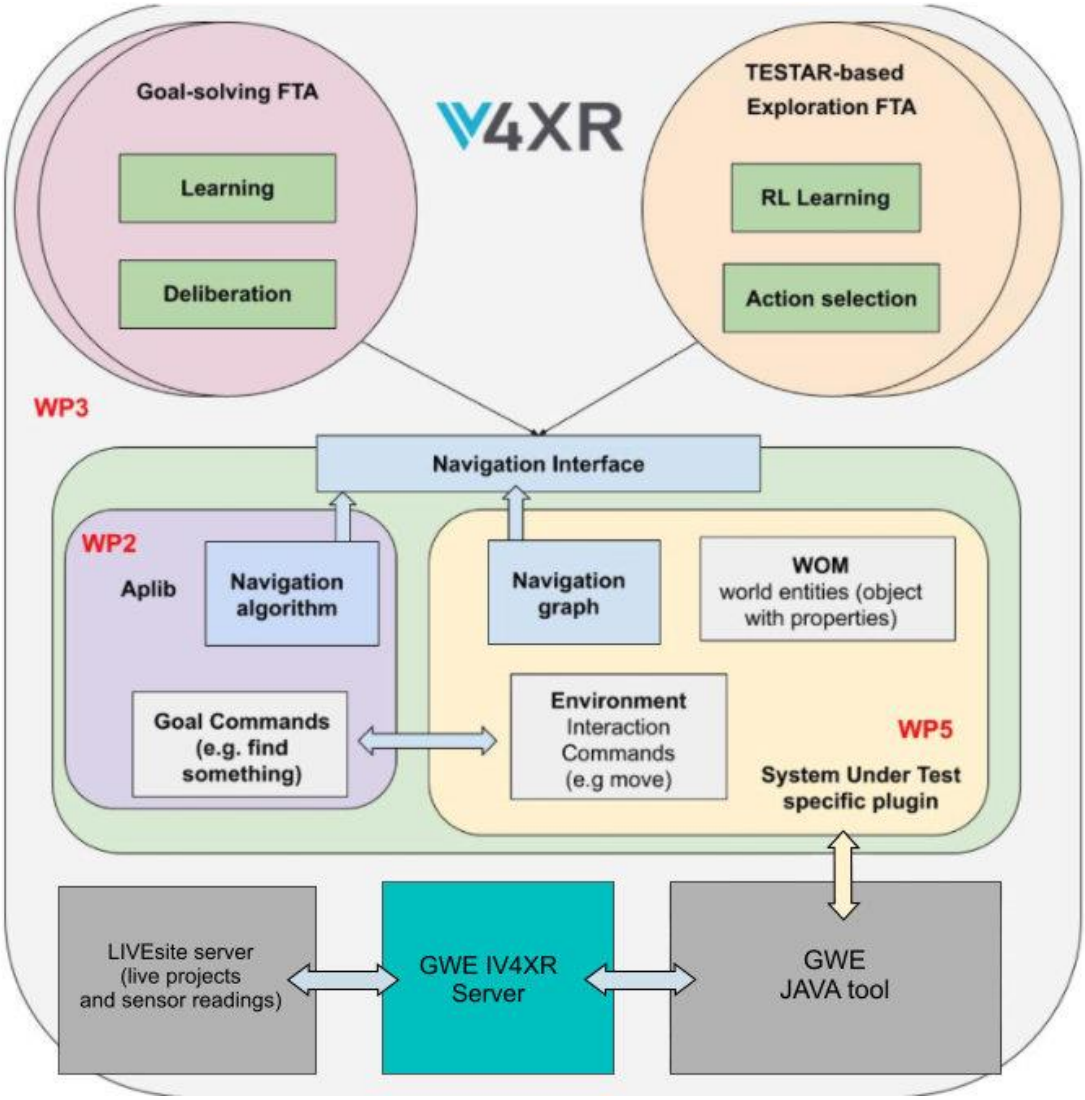
## 6.4 MULTI-SITES

Many engineering monitoring projects are so complex the site is broken down into sub-sites, each of which have their own hosting server which connects to the sensors for that area.

We used the same hierarchical approach for these types of complex sites, to trigger initial agents for each sub-site, each of which can then further analyse the sensors at that sub-site. Simple summaries of errors are then made available, highlighting problems at each sub-site.



## 6.5 Full Integration for Test type 2 - Train moving over structure




The hierarchical approach for test case type 1 is equally valid for trains moving over structures.

For this type of project, the frequency of readings of sensor data is typically increased when the train is detected moving over the bridge, so vibrations and behaviour can be monitored more closely until the train exits the bridge.

For this case our test script is triggered with normal mode, or detail mode, depending on whether the train is on the bridge. In detail mode, the vibrations are checked as a priority, whereby in normal mode, the scripts trigger agents similar to the fixed structure, to verify the overall state of the system and look for errors in sensors and sensor relationships.

Detecting excessive displacements along sections of a bridge is shown below.

## 6.6 FURTHER WORK

So far, all testing has been performed on mirrors of actual monitoring servers, to ensure no disruption to actual monitoring systems happened whilst testing.

The JAVA tool can detect simple sensor errors such as flatlines, jitter, missing values, calibration errors, as well as more complex errors such as formula-based readings being invalid due to one or more errors in any sensor which is used by the formulae.

The agents created can request further tests in more (time-based) detail, as well as record such errors detected so far.

The JAVA tool cannot at present detect errors which could possibly happen in the future due to trends in the data at present. (for instance if a vibration was getting consistently stronger over a

period of time, it could possibly exceed a threshold X days from now, so an advanced warning could be given, rather than waiting for the problem to happen)

We are looking at adding trend-analysis into the tool so that we can spot these kinds of errors. This would help with any kind of monitoring where small changes are occurring over time which are not obvious from graphical readouts, or due to the duration of the changes.

Although tasks are given to the JAVA tool in a hierarchical manner, sometimes a sub-task is a more intensive/time-consuming task than the parent task which requested it (such as checking readings for a sensor every second, instead of every minute).

For this reason we are looking at enhancing how tasks can be broken down, terminated after partial completion, and then restarted, and cancelled. If a sensor is deemed to be invalid for some reason (such as no power going to the hub to which the sensor is connected), checking the sensor in more detail is a waste of bandwidth.

Ensuring tasks take a smaller portion of time means they will be less likely to overload the database with queries, and makes it possible to launch more of the tasks also.

# 7 CONCLUSIONS

This deliverable has presented the progress of WP5 and discussed the full integration of the pilots with the framework. Full integration has been defined as a state at which all relevant features of the pilot system are exposed to the automated testing agents. Each of the pilots have achieved this and some preliminary results of experimentation using the interface has been presented here.

Crucially however, is that while the interfaces expose all essential information, some of the information is not structured optimally for would-be writers of test agents. Therefore the dialogue between the developers and the users continues as quality of life features are added.

The interfaces and development progress discussed here are available on the github page[10] for the project.

[10] https://github.com/iv4xr-project

# A1 Space Engineers Plugin User Manual

This manual will get you started with using **Space Engineers**[11] with the iv4XR framework. A continuously updated copy of this information, and all the code/binaries/assets that are referred herein are located at the repository on github[12]. For more information on the iv4XR project, please refer to iv4xr-project.eu[13].

## 1.1 Introduction

Space Engineers is a sandbox game by Keen Software House. This project is a plugin for the game which enables its integration with the iv4XR testing framework. The plugin runs a TCP/IP server with JSON-RPC API. It allows access to the surrounding of the player's character in a structured form, and to control the characters which are under control of the client. While this plugin is intended to facilitate the intelligent testing of SE using agents, the plugin provides a general purpose api that is suitable for many purposes.


The project also includes a fully-featured client in Kotlin[14].

[11] https://www.spaceengineersgame.com/
[12] https://github.com/iv4xr-project/iv4xr-se-plugin
[13] https://iv4xr-project.eu/
[14] https://github.com/iv4xr-project/iv4xr-se-plugin/tree/main/JvmClient

### 1.2. How to run the game with this plugin

It's not necessary to build anything to try out this plugin. This section describes how to do it.

1. Obtain the binary release of Space Engineers (buy it on Steam or get a key). Install the game.
2. Obtain a binary release of the plugin. Look for the releases[15] section in the repository and for assets of the chosen release. Download the three DLL libraries.
    1. If you want the latest changes or you'd like to edit the code, you can also build it from the sources (even if you don't have the Space Engineers source code), see the section **How to build** below.
3. IMPORTANT: Make sure Windows is OK to run the libraries. **Windows 10 blocks user downloaded libraries by default.** To unblock them, right-click each of them and open file properties. Look for the “Security “section on the bottom part of the General tab. You might see a message: "*This file came from another computer and might be blocked...*". If so, check the Unblock checkbox. If you skip this step, the game will probably crash with a message: System.NotSupportedException: *An attempt was made to load an assembly from a network location...*
4. Obtain other libraries as described in the section **3rd Party Dependencies** below.
5. Put the plugin libraries (and their dependencies) into the folder with the game binaries. A common location is C:\Program Files (x86)\Steam\steamapps\common\SpaceEngineers\Bin64. Tip: You can put the libraries into a subfolder (such as ivxr-debug). Or, it can be a symbolic link to the build folder of the plugin project. In that case, you must prefix the name of each library with ivxr-debug\ in the following step.
6. Right-click on the game title in the Steam library list and open its properties window. Click on the **Set launch options...** button. Add the option “-plugin” followed by the location of the main plugin library. Library dependencies will be loaded automatically – just make sure they are in the same folder or some other searched location. The resulting options line should look something like this: “-plugin Ivxr.SePlugin.dll”.
7. Run the game. (If the game crashes, make sure you've seen step 3.)
8. Start a scenario.
9. If the plugin works correctly, a TCP/IP server is listening for JSON-RPC calls on a fixed port number. (The current development version uses the port number 3333.) Another sign of life is a log file present in user's app data folder such as: C:\Users\<username>\AppData\Roaming\SpaceEngineers\ivxr-plugin.log

#### 1.2.1 3rd Party Dependencies

Apart from the game libraries, the plugin requires two additional libraries to run:

- AustinHarris.JsonRpc.dll, which in turn requires:

[15] https://github.com/iv4xr-project/iv4xr-se-plugin/releases

- Newtonsoft.Json.dll

There are many ways to obtain the libraries. The library DLLs are also a part of the releases page of the repository, but another method is to build them yourself:

- Check-out the JSON-RPC.NET master branch on GitHub[16].
    - *Side note: The binary releases are not updated (compared to NuGet packages), but the last release v1.1.74[17] works as well. You can try it if the master branch does not.*
- Build the solution including the test project (tested with Visual Studio 2019).
- You will find the AustinHarris.JsonRpc.dll library in this path: Json-Rpc\bin\Debug\netstandard2.0
- And the Newtonsoft.Json.dll library in this path: AustinHarris.JsonRpcTestN\bin\Debug\netcoreapp3.0

Note: If you build the project from the sources as described in the section **How to Build**, the libraries are downloaded via NuGet packages in the same way.

### 1.3 How to build

First of all, you don't *have to* build it from sources. The steps above allow you to download the official binary releases which are updated every so often. If you wish to obtain the absolute latest features, then follow these instructions.

The plug-in requires Space Engineers libraries to compile. There are two ways to provide the libraries: as binaries (DLLs) or as sources. Both options are described below.

The resulting plug-in (a couple of .NET libraries) works with the official Steam version of Space Engineers without any modification of the game.

We are developing the plugin using source dependencies; therefore, it is necessary to perform a few steps to switch to binary dependencies: 1.) provide the library binaries and 2.) switch the references to those binaries. We assume you have installed the official release of the game from Steam.

1. **Obtain the Space Engineers libraries.** Locate the script copydeps.bat in the BinaryDependencies directory.
    1. If you have your Steam installation of Space Engineers in the default path, then just run the script and the binaries will be copied to the directory. The default path is C:\Program Files (x86)\Steam\steamapps\common\SpaceEngineers\Bin64.
    2. If you have SE in some other path, provide this path as the first argument to the script. (Or copy the libraries listed in the script manually.)

[16] https://github.com/Astn/JSON-RPC.NET
[17] https://github.com/Astn/JSON-RPC.NET/releases/tag/v1.1.74

2. **Switch references from project dependencies to binary ones.** You can do it manually, or you can apply (cherry-pick) a commit pointed to by the branch binary-deps-switch. (Currently it's commit fa7c536f57, but it can change as we rebase it on newer history.)
3. **Open the solution and build it.** Find the VS solution file Iv4xrPluginBinaryDeps.sln in the Solutions folder. It's just a solution containing only the plugin projects, not the SE projects – the switch to the binaries has to be done in each of the projects, as described in the previous step. Open the solution and build it. You can then run the game with the plugin as described above.\\\\\

## 1.4 API

The network protocol is based on JSON-RPC 2.0[18]. JSON-RPC messages are separated by newlines, TCP is used as the transport layer. The protocol (individual API) is now more stable than in the beginning of the development, but it's still possible it will change as we learn new things.

For an up to date list of provided API calls see the interfaces `ISpaceEngineers`[19] and `ISpaceEngineersAdmin`[20] in the project `Ivxr.SpaceEngineers`.

You can also check out the JvmClient for a client side implementation of the interface in Kotlin and examples how to use it. See also the Usage Example section below.

### 1.4.1 Architecture Overview

Overview of the solution projects:

- `Ivxr.SePlugin` – The **main plugin project**. Contains most of the important logic. It is one of the plugin libraries, the main one.
  - See the project details below.
- `Ivxr.SePlugin.Tests` – Unit tests for the main project.
- `Ivxr.PlugIndependentLib` – Contains service code that is *entirely independent of the Space Engineers codebase* for services such as the logging interface, and classes for configuration and the JSON-RPC library
- `Helpers` – A utilities project which contains mock servers which run a TCP/IP server based on the infrastructure from the two main libraries and using some simple mock implementations of the classes which would normally depend on a running game.

[18] https://www.jsonrpc.org/specification
[19] https://github.com/iv4xr-project/iv4xr-se-plugin/blob/main/Source/Ivxr.SpaceEngineers/ISpaceEngineers.cs
[20]https://github.com/iv4xr-project/iv4xr-se-plugin/blob/main/Source/Ivxr.SpaceEngineers/ISpaceEngineersAdmin.cs

### 1.4.2 PROJECT DETAILS: IVXR.SEPLUGIN

List of notable classes – top level:

- `IvxrPlugin` – Entry point of the plugin, implements the game's IPlugin interface.
- `IvxrSessionComponent` – Inherits from game's MySessionComponentBase which allows the component to hook the plugin into game events such as UpdateBeforeSimulation called each timestep of the game.
- `IvxrPluginContext` – Root of the dependency tree of the plugin, constructs all the important objects.

Notable sub-namespaces (and the solution sub-folders):

- `Control` – Interfacing with the game: Obtaining observations and controlling the character. Notable classes:
  - `Dispatcher` – The command hub.
  - `CharacterController` – Self-explanatory.
  - `Observer` – Extracts observations from the game.
  - `SessionController` – Session control such as loading a saved game.
- `JSON` – A wrapper around the JSON providers .
- `Navigation` - Classes to support the automated navigation of the agent around grids.

## 1.5 SPACE ENGINEERS ENGINE INFORMATION

This section describes how the SE engine and usints work within the game, in this section we will go over sizes, positions, movement, blocks, and multiple characters. A continuously updated version of this document can be found on the github repository page here[21]. The API listing that will be referred to in this section can be found here[22].

### 1.5.1 UNITS AND POSITION

- One big block size is 2.5 game meters. Some vectors are sent in meters, some are sent in "cubes".
- Block size (`Block.size`) is in large cubes so for example a 1x1x1 large block is 2.5 x 2.5 x 2.5 meters.
- Use the enum `CubeSize` and/or the constants `LARGE_BLOCK_CUBE_SIDE_SIZE`, `SMALL_BLOCK_CUBE_SIDE_SIZE` for conversions.
- 5 small blocks to one big block. (So a small block cube is 0.5x0.5x0.5 meters.)
- The engineer character can fit into space of 2x3x2 in small blocks (1x1.5x1 meters), however the size in the code is 1x1.8x1.

[21] https://github.com/iv4xr-project/iv4xr-se-plugin/tree/main/JvmClient
[22] https://iv4xr-project.github.io/iv4xr-se-plugin/index.html

- The position of character is at its bottom, the camera is not. The offset vector between the bottom of the character and the camera is: (x=0, y=1.6369286, z=0). Use the `Character.DISTANCE_CENTER_CAMERA` constant (or you can use the difference between position and camera position).
- Block position is always between `minPosition` and `maxPosition`, but it doesn't always have to be in the center of the block (or sometimes it's identical to `minPosition`). To locate the center of the block, use the midway point between `minPosition` and `maxPosition` (or the extension function `centerPosition`).

### 1.5.2 Character and camera orientation

- Character forward vector is identical to the camera forward vector - when moving to a side, both forward vectors are changed.
- Character up vector differs from the camera up vector when walking. Imagine a character moving his head to look up rather than whole body.
- When the jetpack is on, up vectors are identical. Imagine a character rotating whole body to look up.
- This works for 3d person camera mode, unknown for other modes.
- There is a possibility to move the camera around the character. What is happening with internal variables is not explored.

### 1.5.3 Basic movement API

- The Method `moveAndRotate` accepts a movement vector. The vector represents a direction.
- It's value defines the type of movement based on the speed. If it's less than 0.4, it is slow movement. If less than or equal to 1.6, it is walking. If over 1.6, it is sprinting. This is relevant when actually walking (ex. not using the jetpack).
- Use the convenience extension methods `normalizeAsWalk`, `normalizeAsRun`, normalizeAsSprint to adjust the vector size to your needs.
- Check `CharacterMovementType` for more information and to check constants.
- There is also movement while in crouch.

### 1.5.4 Movement types and speed

There are other movement types for Space Engineers, so this is not a full list of possibilities. Following table describes differences between movements and their speeds.

| Movement type | Max speed (m/s) | Movement vector threshold |
|---|---|---|
| Crouch walk | 2 | ? |
| Walk | 3 | 0.4 < |
| Run | 6 | 1.6 <= |
| Sprint | 10 | 1.6 > |
| Jetpack | 110 | ? |

### 1.5.5 CONTINUOUS MOVEMENT

Calling `moveAndRotate` will behave in a similar fashion as a single keyboard stroke. To keep moving, the command has to be sent repeatedly, behaving as a key constantly being pressed. This is quite inconvenient for the code and not very deterministic since commands are sent rapidly over TCP without any kind of time synchronization.

For that reason, `moveAndRotate` has a `ticks` parameter with the default value of 1, which determines the number of ticks for the command to be active (equivalent to the key being pressed). One second has 60 ticks. To stop the movement preemptively before the specified number of ticks elapses, call `moveAndRotate` with 0 ticks (or supersede the movement by sending a new command).

This movement is quite deterministic when repeating exactly the same scenario with exact positions and movement values. Sometimes the values are slightly off however, especially when the scenario is loaded for the first time.

### 1.5.6 BLOCKS

Blocks are a core concept in the game. Some data and behaviour is documented to avoid confusion. Some of these concepts are based on observations and are not verified by code so if you find inconsistencies in anything, please raise an issue[23].

[23] https://github.com/iv4xr-project/iv4xr-se-plugin/issues

#### 1.5.6.1 DEFINITIONID, ID

The Block type itself is a unique combination of id and type (now represented by the `DefinitionId` class). Blocks are the same type, if they have the same `DefinitionId`.

Created block instances have their own ID (this is a different id from the block `DefinitionId`). This ID is a supposedly unique random integer. The current implementation exposes an internal block ID, and the exact guarantees of uniqueness are not currently known – we expect it to be unique at least within the grid, hopefully within the whole scenario – please report any collisions.

#### 1.5.6.2 BLOCK-SPECIFIC INSTANCE PROPERTIES

Basic properties that are common for all block types are defined in the interface `Block`. In this sense, we are talking about mutable properties, which can have different values for every instance of a block (such as `position` or `integrity`).

Some blocks have extra properties related to their functionality. For example, all door blocks have a boolean `opened` property. Doors in combination with `use` functionality can allow the client to open a door and check the door state.

Generators have a property which represents output, etc. Functional and Terminal blocks are very common.

When calling via the JSON API, you always get an instance of `Block`, but if blocks have extra properties, it will be a subclass with these extra properties. The returned list is polymorphic and each block has an instance depending on its type. You can use standard Java/Kotlin keywords like `instanceof` or `is` to check for subclasses and cast. Check `PolymorphicBlocksTest` for a Kotlin example.

The class hierarchy is not flat. You can explore the whole block hierarchy using the `Definitions.BlockHierarchy` API, which returns relations between blocks. Subclassing is done only for classes which have extra properties. Otherwise, only `Block` is returned (or very often `FunctionalBlock` or `TerminalBlock`). Information about class hierarchy and what classes will be used for different blocks is also cached in `src/jvmMain/resources` directory as JSON files.

If there is a property or a block type functionality missing in the API, and you'd like it to be added, please file an issue to let us know.

#### 1.5.6.3 DEFINITIONS

Each block type has its own definition properties, which contains basic immutable properties shared among all blocks of the same type. To list all available definitions, call `Definitions.BlockDefinitions`. Individual properties are documented in the code of BlockDefinition class.

#### 1.5.6.4 SMALL VS LARGE CUBE BLOCKS

Most of the blocks are cube blocks, which have cube size 1x1x1. Cube blocks can be small or big. Small blocks have 0.5 meters. Big blocks have 2.5 meters, so 5 times bigger, 125 small blocks fit into one big block.

Large blocks usually have the `Large` prefix, their small counterparts have the `Small` prefix.

#### 1.5.6.5 TARGETING A BLOCK

When trying to grind a block down or weld it up, there are a few things to keep in mind:

- `CharacterObservation` has the property `targetBlock`. This property is not null only if there is currently a block in front of the character in close range and the cursor is pointing directly at it.
- Different grinders and welders have different ranges so distance from the block matters for different tools.
- Sometimes targeting is not as simple as walking up to the block. The cursor has to directly target the model and if the model has for example hollow places (like with an open door), if you point at the hollow space, you are not targeting the block. This is especially tricky when trying to target blocks automatically via a script.
- In rare cases targeting the block model doesn't work either (which can happen with the models for wheels), and it requires some moving around. This appears to be a glitch in the game.

One way to fix this targeting issue is to try to move the cursor around a bit until `targetBlock` is the desired block. Another approach is to try to use mount points. Each block has mount points, which are areas that can be used to connect to other blocks. Getting to the position of the mount point has a higher chance of that part being solid and flat and to successfully target the block. Iterating over all mount points increases the chance even further. This still doesn't guarantee success in all cases, but works for most. There are extension methods to calculate mount point positions and orientations of blocks such as `mountPointToRealWorldPosition`.

#### 1.5.6.6 USING BLOCKS

A block can have multiple `useObjects` – active parts that can do something when used. For instance, it can be a chair, on which you can sit. But door blocks have both the door themselves, and a terminal. Button blocks can have up to 4 buttons and each button is different. Information about block's use objects is in the `useObjects` property.

It is possible to "use" blocks by emulating the "F" key in the game. To do that, use the `Character.Use` API call.

This requires that the cursor is pointing exactly at the active part of the block to work. If there's an active use object, it will be in `CharacterObservation.TargetUseObject`.

There is currently no reliable way to point at specific use objects, but there is also an admin command that allows activating the `useObjects` of blocks without the need to precisely target them. `Admin.Character.Use`

#### 1.5.6.7 COMPOUND BLOCKS

Building some blocks actually doesn't build a single block, but a pair of blocks that are tightly bound together. They behave as normal 2 separate blocks.

If block A spawns blocks A and B together, trying to build block B by itself either:

- Doesn't work at all.
- Spawns only block B.

At least how it was tested so far.

That is normal game behaviour. When placing a block programmatically using the admin command `placeAt`, only a single block is created and it is therefore possible to place these secondary blocks alone, even though it is not possible normally.

### 1.5.7 WELDING AND GRINDING

Using different welders and grinders produces different results. They have different speeds and reaches. Below is a formula that tries to help with determining grinding/welding speed.

### SHARED CONSTANTS:

`ToolCooldownMs = 250`

| Welder | SpeedMultiplier | DistanceMultiplier |
|---|---|---|
| Welder | 1 | 1 |
| Welder2 | 1.5 | 1.2 |
| Welder3 | 2 | 1.4 |
| Welder4 | 5 | 1.6 |

`WELDER_AMOUNT_PER_SECOND = 1` - This is a constant
`WelderSpeedMultiplier = 2` - This is the default value. It is configurable in the scenario between 0.5 and 5

### Final formula:

```
WeldingSpeed = WelderSpeedMultiplier * SpeedMultiplier *
WELDER_AMOUNT_PER_SECOND * ToolCooldownMs / 1000.0
```

### ANGLE GRINDER CONSTANTS AND FORMULA

| Grinder | SpeedMultiplier | DistanceMultiplier |
|---|---|---|
| AngleGrinder | 1 | 1 |
| AngleGrinder2 | 1.5 | 1.2 |
| AngleGrinder3 | 2 | 1.4 |
| AngleGrinder4 | 5 | 1.6 |

`GRINDER_AMOUNT_PER_SECOND = 2` - This is a constant
`GrinderSpeedMultiplier = 2` - This is the default value. It is configurable in the scenario

**Final formula:**

```
GrindingSpeed = GrinderSpeedMultiplier * SpeedMultiplier *
GRINDER_AMOUNT_PER_SECOND * ToolCooldownMs / 1000.0
```

### 1.5.8 Advanced/plugin customization - Adding new block-specific fields

Since the code is polymorphic and everything is static on the code level, we generate the code based on configuration and source files. All the code is generated by `BlockMappingGeneratorRunner`, and it generates both the code for C# side and Kotlin side. It generates classes for `Block`, `BlockDefinitions` and their mappings and serializers. Generating blocks and block definitions are very similar, defining fields and interfaces is the same for both.

Mappings for the `Block` interface is written manually on the C# side and not generated, because we often manually retrieve data and convert to different structures. Mappings for `BlockDefinitions` are done automatically, because we expect it to be simple field pass.

A few tips:

- Commit or stash your changes so that you can easily revert changes in case generated code does not compile.
- If the code doesn't compile after running the generator, find out why, fix, revert the code, then run the generator again.

#### Base interface

To add new field to the `Block` interface:

- Add the field to the `commonBlockFields` variable in `BlockMappingGenerator`.
- Run the generator.
- Add the field to the `Block` interface itself.

#### Polymorphic blocks

To add or modify a polymorphic block type:

- Add/modify a record in `spaceEngineers.controller.blockMappings`
- Run the generator.

### 1.5.9 Multiple characters

Most of the plugin commands by default control the main local character, which is also bound to the camera and input controls. It is possible to add more characters into the game by using

`Admin.Character.Create`. To switch between characters, use `Admin.Character.Switch`, passing the character ID. The main character ID is always `se0`.

A few points:

- This feature is a bit experimental, there may be glitches and unexpected behaviour, please file an issue if you run into anything.
- The toolbar is shared between characters.
- Only the main character can place blocks normally (but any character can place blocks through admin commands).

#### COMMANDS UNAFFECTED BY CHARACTER SWITCH

- `Session.LoadScenario`
- `Observer.TakeScreenshot`
- `Blocks.Place` (always places for the main character - use `Admin.Blocks.PlaceInGrid` / `PlaceAt` to place blocks for other characters)

### 1.6 PLUGIN EXAMPLES

The plugin can be used either directly or via the iv4XR framework with minor changes. The JVM client written in Kotlin directly uses the API of the plugin to effect changes in the game. The plugin itself is also integrated into the iv4XR framework to be used by external agents and systems such as TESTAR. This section has a couple of annotated examples in Kotlin using the different methods. These examples can be found in the repository[24][25].

**Using the iv4XR framework:**

```
package spaceEngineers.iv4xr

import environments.SeAgentState
import environments.SeEnvironment
import eu.iv4xr.framework.mainConcepts.TestAgent
import eu.iv4xr.framework.mainConcepts.TestDataCollector
import eu.iv4xr.framework.spatial.Vec3
import nl.uu.cs.aplib.AplibEDSL.SEQ
import nl.uu.cs.aplib.mainConcepts.GoalStructure
import org.junit.jupiter.api.Disabled
import org.junit.jupiter.api.Test
import spaceEngineers.controller.*
import spaceEngineers.controller.SpaceEngineers.Companion.DEFAULT_AGENT_ID
import spaceEngineers.iv4xr.goal.GoalBuilder
import spaceEngineers.iv4xr.goal.TacticLib
import spaceEngineers.model.ToolbarLocation
import kotlin.test.assertTrue
```

[24] https://github.com/iv4xr-project/iv4xr-se-plugin/blob/main/JvmClient/src/jvmTest/kotlin/spaceEngineers/game/BasicUsageTest.kt
[25] https://github.com/iv4xr-project/iv4xr-se-plugin/blob/main/JvmClient/src/jvmTest/kotlin/spaceEngineers/iv4xr/BasicIv4xrTest.kt

```
class BasicIv4xrTest {

    @Disabled("Disabled for building whole project, enable manually by uncommenting.")
    @Test
    fun placeGrindDownTorchUp() {
        // Setup constants to use later.
        val agentId = DEFAULT_AGENT_ID //agent id
        val blockType = "LargeHeavyBlockArmorBlock" // Block type that we will operate with (it's a cube block).
        val context = SpaceEngineersTestContext() // This context saves recent information about operations (for
example last built blocks and all the observations).
        val blockLocation = ToolbarLocation(1, 0) // We will put the block here in the toolbar.
        val l welder = "Welder2Item" // We will use this welder.
        val welderLocation = ToolbarLocation(2, 0) // We will put the welder here in the toolbar.
        val grinder = "AngleGrinder2Item" // We will use this grinder.
        val grinderLocation = ToolbarLocation(3, 0) // We will put the grinder here in the toolbar.
        // We map the position of the block in the toolbar.
        context.blockTypeToToolbarLocation[blockType] = blockLocation
        // We create an instance of the SpaceEngineers interface. ContextControllerWrapper is a "smarter"
implementation that saves recent information into the context created above.
        // Otherwise, JsonRpcSpaceEngineersBuilder.localhost(agentId) can be used directly (also SpaceEngineers
interface implementation).
        val controllerWrapper =
            ContextControllerWrapper(
                spaceEngineers = JsonRpcSpaceEngineersBuilder.localhost(agentId),
                context = context
            )


        // We create the iv4xr environment and pass the ID of the world (scenario to load).
        val theEnv = SeEnvironment(
            controller = controllerWrapper,
            worldId = "simple-place-grind-torch-with-tools",
        )

        // Creating IV4XR related classes.
        val dataCollector = TestDataCollector()

        val myAgentState = SeAgentState(agentId = agentId)

        // Assemble agent.
        val  testAgent = TestAgent(agentId, "some role name, else nothing")
            .attachState(myAgentState)
            .attachEnvironment(theEnv)
            .setTestDataCollector(dataCollector)

        val goals = GoalBuilder()
        val tactics = TacticLib()
        // Create goals and tactics.
        val testingTask: GoalStructure =  SEQ(
            goals.agentAtPosition(Vec3(532.7066f, -45.193184f, -24.395466f), epsilon = 0.05f),
            goals.agentDistanceFromPosition(
                Vec3(532.7066f, -45.193184f, -23.946253f),
                distance = 16f,
                epsilon = 0.1f,
                tactic = tactics.moveForward(),
            ),
            goals.blockOfTypeExists(
                blockType,
                tactic = tactics.buildBlock(blockType),
            ),
```

```
            goals.lastBuiltBlockIntegrityIsBelow(
                percentage = 0.1,
                tactic = SEQ(
                    tactics.equip(grinderLocation),
                    tactics.sleep(500),
                    tactics.startUsingTool(),
                ),
            ),
            goals.alwaysSolved(
                tactic = SEQ(
                    tactics.endUsingTool(),
                    tactics.sleep(500),
                ),
            ),
            goals.lastBuiltBlockIntegrityIsAbove(
                percentage = 1.0,
                tactic = SEQ(
                    tactics.equip(welderLocation),
                    tactics.sleep(500),
                    tactics.startUsingTool(),
                ),
            ),
            goals.alwaysSolved(
                tactic = SEQ(
                    tactics.endUsingTool(),
                ),
            ),
        )

        testAgent.setGoal(testingTask)

        // We load the scenario.
        theEnv.loadWorld()
        // Setup block in the toolbar.
        controllerWrapper.items.setToolbarItem(blockType, blockLocation)
        // Setup welder in the toolbar.
        controllerWrapper.items.setToolbarItem(welder, welderLocation)
        // Setup grinder in the toolbar.
        controllerWrapper.items.setToolbarItem(grinder, grinderLocation)
        Thread.sleep(500)

        // We observe for new blocks once, so that current blocks are not going to be considered "new".
        theEnv.observeForNewBlocks()

        // Run the agent and update in the loop.
        var i = 0
        while (testingTask.status.inProgress() && i <= 1500) {
            testAgent.update()
            println("*** $i, ${myAgentState.wom.agentId} @${myAgentState.wom.position}")
            i++
        }

        // Print results.
        testingTask.printGoalStructureStatus()
        testingTask.subgoals.forEach { assertTrue(it.status.success()) }
    }
}
```

**Using the iv4XR framework as a plugin by external agents:**

The TESTAR tool is a FTAs focus on exploring the virtual environment, it generates test sequences of (state,action)-pairs by connecting to the SE system in its initial state and continuously selecting an action to bring the SUT in another state and check oracles. *Deliverable 3.3 - 2nd prototype of Functional Test Agents (FTAs)* contains more details about TESTAR.

TESTAR integrates the iv4XR framework as a Java plugin[26] to be able to connect and launch SE levels[27], obtain the information of the observable entities and their properties to create the TESTAR State[28], and execute TESTAR actions[29] to interact with the SE environment.

```
package eu.testar.iv4xr.se;

import spaceEngineers.controller.ContextControllerWrapper;
import spaceEngineers.controller.JsonRpcSpaceEngineers;
import spaceEngineers.controller.JsonRpcSpaceEngineersBuilder;
import spaceEngineers.controller.SpaceEngineersTestContext;
import uuspaceagent.UUSeAgentState;

…

public class SpaceEngineersProcess extends SUTBase {

…

private SpaceEngineersProcess(String path, boolean processListenerEnabled) {
  Assert.notNull(path);

  // regex to split checking last space
  String[] parts = path.split(" (?!.* )");

  // If SUTConnectorValue is not correct throw an informative error
  if(parts.length < 1 || parts.length > 2 ) {
    settingsConnectorError();
  }

  // Prepare SUTConnectorValue parts to launch SpaceEngineers and launch the desired level
  String launchPart = parts[0].trim();
  String levelPath = "";
  if(parts.length == 2) {
    levelPath = parts[1].replace("\"", "");
  }

  // Launch and connect with SpaceEngineers
  launchSpaceEngineers(launchPart, processListenerEnabled);
```

[26] https://github.com/iv4xr-project/TESTAR_iv4xr/tree/v3.2/testar/lib

[27] https://github.com/iv4xr-project/TESTAR_iv4xr/blob/v3.2/iv4XR/src/eu/testar/iv4xr/se/SpaceEngineersProcess.java

[28] https://github.com/iv4xr-project/TESTAR_iv4xr/blob/v3.2/iv4XR/src/eu/testar/iv4xr/se/SeStateFetcher.java

[29] https://github.com/iv4xr-project/TESTAR_iv4xr/blob/v3.2/iv4XR/src/eu/testar/iv4xr/actions/se/commands/seActionCommandPlaceBlock.java

```
  Util.pause(5);

  if(!win.isRunning()) {
    throw new SystemStartException(String.format("ERROR trying to connect with SE iv4xr SUT : %s", launchPart));
  }

  /**
   * Start IV4XR SUT at WOM level
   */
  try {
    // Prepare SpaceEngineers Controller
    JsonRpcSpaceEngineers seRpcController = JsonRpcSpaceEngineersBuilder.Companion.localhost(characterControllerId);
    Util.pause(2);
    System.out.println("Welcome to the SE iv4XR test: " + launchPart);

    // Load Space Engineers Level
    if(!levelPath.isEmpty()) {
      seRpcController.getSession().loadScenario(new File(levelPath).getAbsolutePath());
      Util.pause(10);
      System.out.println("Loaded level: " + levelPath);
    }

    // Create UU state grid
    UUSeAgentState stateGrid = new UUSeAgentState(characterControllerId);
    // Prepare UU agent
    SpaceEngineersTestContext context = new SpaceEngineersTestContext();
    ContextControllerWrapper controllerWrapper = new ContextControllerWrapper(seRpcController, context);
    // WorldId is empty because we are going to connect to a running level, not load a new one
    SeEnvironment sEnv = new SeEnvironment("", controllerWrapper, context);
    // Finally create the TestAgent
    TestAgent testAgent = new SeAgentTESTAR(characterControllerId, "explorer").attachState(stateGrid).attachEnvironment(sEnv);

    this.set(IV4XRtags.windowsProcess, win);
    this.set(Tags.PID, win.pid());
    this.set(IV4XRtags.iv4xrSpaceEngRpcController, seRpcController);
    this.set(IV4XRtags.iv4xrSpaceEngCharacter, seRpcController.getCharacter());
    this.set(IV4XRtags.iv4xrSpaceEngItems, seRpcController.getItems());
    this.set(IV4XRtags.iv4xrTestAgent, testAgent);

  } catch(Exception e) {
    System.err.println(String.format("EnvironmentConfig ERROR: Trying to connect with %s", launchPart));
    System.err.println(e.getMessage());
    win.stop();
    throw new SystemStartException(e);
  }
  iv4XR = this;
}
```

```
package eu.testar.iv4xr.se;

import spaceEngineers.controller.JsonRpcSpaceEngineers;
import spaceEngineers.controller.Observer;
import spaceEngineers.model.CubeGrid;
import spaceEngineers.model.Observation;
import spaceEngineers.model.Block;
import spaceEngineers.model.CharacterObservation;
…

public class SeStateFetcher extends IV4XRStateFetcher {
```

```
…

/**
 * Create an Array tree of elements that later becomes the Widget-tree.
 * Use the Space Engineers Rpc Controller to extract the Agent and Blocks information from the WOM.
 * Every instant of time the Agent will observe himself and the Blocks if these are in close range.
 */
@Override
protected IV4XRRootElement fetchIV4XRElements(IV4XRRootElement rootElement) {
  // Get the controller attached to the SE system (SpaceEngineersProcess)
  JsonRpcSpaceEngineers seRpcController = system.get(IV4XRtags.iv4xrSpaceEngRpcController);

  // Check that TESTAR it can observe the SE system
  Observer seObserver = seRpcController.getObserver();
  if(seObserver == null) {
    throw new StateBuildException("SE Agent Oberver is null! Exception trying to fetch the State of iv4XR SpaceEngineers");
  }

  // Get the Character and Blocks observation that we use to create the element tree
  CharacterObservation seObsCharacter = seRpcController.getObserver().observe();
  Observation seObsBlocks = seRpcController.getObserver().observeBlocks();

  // If the agent observes himself and in this instant of time also has observation of blocks
  if(seObsCharacter != null && seObsBlocks != null && seObsBlocks.getGrids() != null && seObsBlocks.getGrids().size() > 0) {
    // Add manually the Agent as an Element (Observed Blocks + 1)
    rootElement.children = new ArrayList<IV4XRElement>((int) seObserver.observeBlocks().getGrids().size() + 1);
    rootElement.zindex = 0;
    fillRect(rootElement);

    // Create the Agent as element of the tree, because always exists as a Widget
    SEagent(rootElement, seObsCharacter);

    // If the Agent observes blocks create the elements blocks tree
    if(seObsBlocks.getGrids().size() > 0) {
      for(CubeGrid seCubeGrid : seObsBlocks.getGrids()) {
        SEGridDescend(rootElement, seCubeGrid);
      }
    }
  }
  // If agent observes himself but in this instant has NO observation of blocks
  else if (seObsCharacter != null) {
    // Add manually the Agent as an Element (Observed Entities + 1)
    rootElement.children = new ArrayList<IV4XRElement>(1);

    rootElement.zindex = 0;
    fillRect(rootElement);
    SEagent(rootElement, seObsCharacter);
  } else {
    System.err.println("ERROR: No Agent and no BLOCKS in the current Observation");
  }
return rootElement;
}

/**
 * Based on the Space Engineers CharacterObservation, extract the agent properties to
 * create an element inside the fetched tree.
 *
 * @param parent
 * @param seObsCharacter
 * @return agent Element
```

```
 */
private IV4XRElement SEagent(IV4XRElement parent, CharacterObservation seObsCharacter) {
  IV4XRElement childElement = new IV4XRElement(parent);
  parent.children.add(childElement);
  childElement.zindex = parent.zindex +1;

  childElement.agentPosition = new Vec3(seObsCharacter.getPosition().getX(), seObsCharacter.getPosition().getY(),
seObsCharacter.getPosition().getZ());
  childElement.seAgentPosition = seObsCharacter.getPosition();
  childElement.seAgentOrientationForward = seObsCharacter.getOrientationForward();
  childElement.seAgentOrientationUp = seObsCharacter.getOrientationUp();
  childElement.seAgentHealth = seObsCharacter.getHealth();

  childElement.entityVelocity = new Vec3(seObsCharacter.getVelocity().getX(), seObsCharacter.getVelocity().getY(),
seObsCharacter.getVelocity().getZ());
  childElement.entityId = system.get(IV4XRtags.iv4xrSpaceEngRpcController).getAgentId();
  childElement.entityType = "AGENT";

  fillRect(childElement);
  return childElement;
}

/**
 * Based on the Space Engineers blocks Observation, extract the CubeGrid properties to
 * create a grid element inside the fetched tree.
 * Then extract the Block that are children of these CubeGrid.
 *
 * @param parent
 * @param seCubeGrid
 * @return CubeGrid element with Block children
 */
private IV4XRElement SEGridDescend(IV4XRElement parent, CubeGrid seCubeGrid) {
  IV4XRElement childElement = new IV4XRElement(parent);
  parent.children.add(childElement);
  childElement.zindex = parent.zindex +1;

  childElement.entityPosition = new Vec3(seCubeGrid.getPosition().getX(), seCubeGrid.getPosition().getY(),
seCubeGrid.getPosition().getZ());
  childElement.entityId = seCubeGrid.getId();
  childElement.entityType = seCubeGrid.getId().replaceAll("[0-9]","").replaceAll("\\s+","");

  fillRect(childElement);

  for(Block seBlock : seCubeGrid.getBlocks()) {
    SEBlockDescend(childElement, seBlock);
  }

  return childElement;
}

/**
 * Extract the Block properties to create a block element inside the fetched tree.
 *
 * @param parent
 * @param seBlock
 * @return Block element
 */
private IV4XRElement SEBlockDescend(IV4XRElement parent, Block seBlock) {
  IV4XRElement childElement = new IV4XRElement(parent);
  parent.children.add(childElement);
  childElement.zindex = parent.zindex +1;
```

```
 childElement.entityPosition = new Vec3(seBlock.getPosition().getX(), seBlock.getPosition().getY(), seBlock.getPosition().getZ());
 childElement.entityId = seBlock.getId();
 childElement.entityType = seBlock.getDefinitionId().getType();

 childElement.seBuildIntegrity = seBlock.getBuildIntegrity();
 childElement.seIntegrity = seBlock.getIntegrity();
 childElement.seMaxIntegrity = seBlock.getMaxIntegrity();
 childElement.seMaxPosition = seBlock.getMaxPosition();
 childElement.seMinPosition = seBlock.getMinPosition();
 childElement.seOrientationForward = seBlock.getOrientationForward();
 childElement.seOrientationUp = seBlock.getOrientationUp();
 childElement.seSize = seBlock.getSize();

 fillRect(childElement);
 return childElement;
}
}
```

```
package eu.testar.iv4xr.actions.se.commands;

import spaceEngineers.model.ToolbarLocation;
…

public class seActionCommandPlaceBlock extends seActionCommand {

…

public seActionCommandPlaceBlock(Widget w, String agentId, String blockType){
  this.agentId = agentId;
  this.blockType = blockType;
  this.set(Tags.OriginWidget, w);
  this.set(Tags.Role, iv4xrActionRoles.iv4xrActionCommandInteract);
  this.set(Tags.Desc, toShortString());
}

@Override
public void run(SUT system, State state, double duration) throws ActionFailedException {
  spaceEngineers.controller.Items seItems = system.get(IV4XRtags.iv4xrSpaceEngItems);
  spaceEngineers.controller.JsonRpcSpaceEngineers seRpcController = system.get(IV4XRtags.iv4xrSpaceEngRpcController);

  seItems.setToolbarItem(blockType, ToolbarLocation.Companion.fromIndex(1, 2));
  seItems.equip(ToolbarLocation.Companion.fromIndex(1, 2));
  seRpcController.getBlocks().place();
}
```

### Using the API directly

```
package spaceEngineers.game

import org.junit.jupiter.api.Disabled
import spaceEngineers.controller.JsonRpcSpaceEngineersBuilder
import spaceEngineers.controller.SpaceEngineers.Companion.DEFAULT_AGENT_ID
import spaceEngineers.controller.loadFromTestResources
import spaceEngineers.model.DefinitionId
import spaceEngineers.model.Vec3F
import spaceEngineers.model.extensions.allBlocks
import testhelp.SIMPLE_PLACE_GRIND_TORCH
import kotlin.test.Test
import kotlin.test.assertEquals
import kotlin.test.assertTrue


class BasicUsageTest {


    /**
     * This scenario:
     * - Teleports the character to a location.
     * - Creates a block there using the admin placeAt function and checks the results.
     * - Removes the block again using admin remove and checks that the block is gone.
     */
    @Disabled("Disabled for building whole project, enable manually by uncommenting.")
    @Test
    fun placeBlock() {
        // We create a SpaceEngineers interface that connects to the  local Space Engineers game.
        val context = JsonRpcSpaceEngineersBuilder.localhost(DEFAULT_AGENT_ID)
        with(context) {
            // We load the testing scenario.
            session.loadFromTestResources(SIMPLE_PLACE_GRIND_TORCH)
            // Create the  ID of the block that we are gonna build.
            val blockDefinitionId = DefinitionId.cubeBlock("LargeBlockSmallGenerator")
            val z = 1000
            // We teleport the character to a specific location and orientation.
            admin.character.teleport(Vec3F(0, 0, z + 15), Vec3F.FORWARD, Vec3F.UP)
            // We observe for new blocks to ensure all the existing blocks won't be "new" anymore.
            observer.observeNewBlocks()
            // We place a block.
            val blockId =
                admin.blocks.placeAt(blockDefinitionId, Vec3F(0, 0, z + 0), Vec3F.FORWARD, Vec3F.UP)
            // We observe for new blocks.
            val block = observer.observeNewBlocks().allBlocks.first()
            // We expect block to be there and to have all the expected properties:
            assertEquals(block.id, blockId)
            assertEquals(block.definitionId.type, block.definitionId.type)
            assertTrue(observer.observeBlocks().allBlocks.any { it.definitionId.type == blockDefinitionId.type })
            assertEquals(12065.0f, block.integrity)
            assertEquals(block.maxIntegrity, block.integrity)
            // We remove the block.
            admin.blocks.remove(block.id)
            // We expect the block to be removed now.
            assertTrue(observer.observeBlocks().allBlocks.none { it.definitionId.type == blockDefinitionId.type })
        }
    }
}
```

# A2 NUCLEAR PLANT INTRUSION SIMULATION USER MANUAL

## 2.1 INTRODUCTION

For confidential reasons, just the code of the plugin of the pilot could be uploaded on GitHub and the user manual, that will be described below, concerns only the preliminary tests of the connections to the pilot through this plugin without going into detail concerning the functioning of the CGF MAEV.

The IT resources used to carry out the tests are as follows:

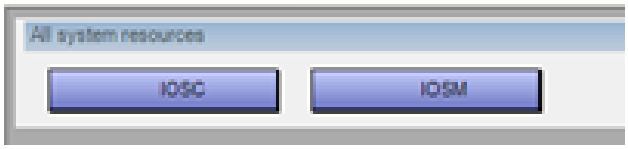
- EFBox : VM « TTSOSNV3DDEV09P » under Windows 7
- PIW-Server 4.5.10, PIW-RPC 13.2.0 (with patch)

## 2.2 TESTS PROCEDURE

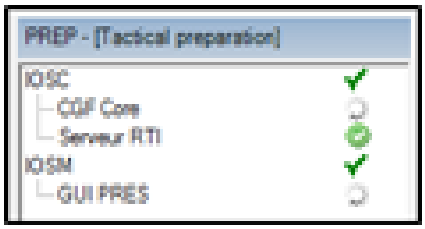
1. Via the PIW tool, with a "**PTF-CEI-Standalone**" platform containing a resource with the type "STANDALONE", install and select the assembly :
   → "**assy-TF-IV4XR**" (It contains the project assembly "**assy-TF-IV4XR**")

2. Run the script "D:\PLTFPIW\ProgramFiles\SystemConfigEnv_TF_Standalone.bat" to prepare the test environment (FRSH, "SimonServers" plastron, etc.).
   Then launch the script " D:\PLTFPIW\ProgramFiles\Start_CONF_Standalone.bat ".

3. The CONF application launches and presents the following system resources:

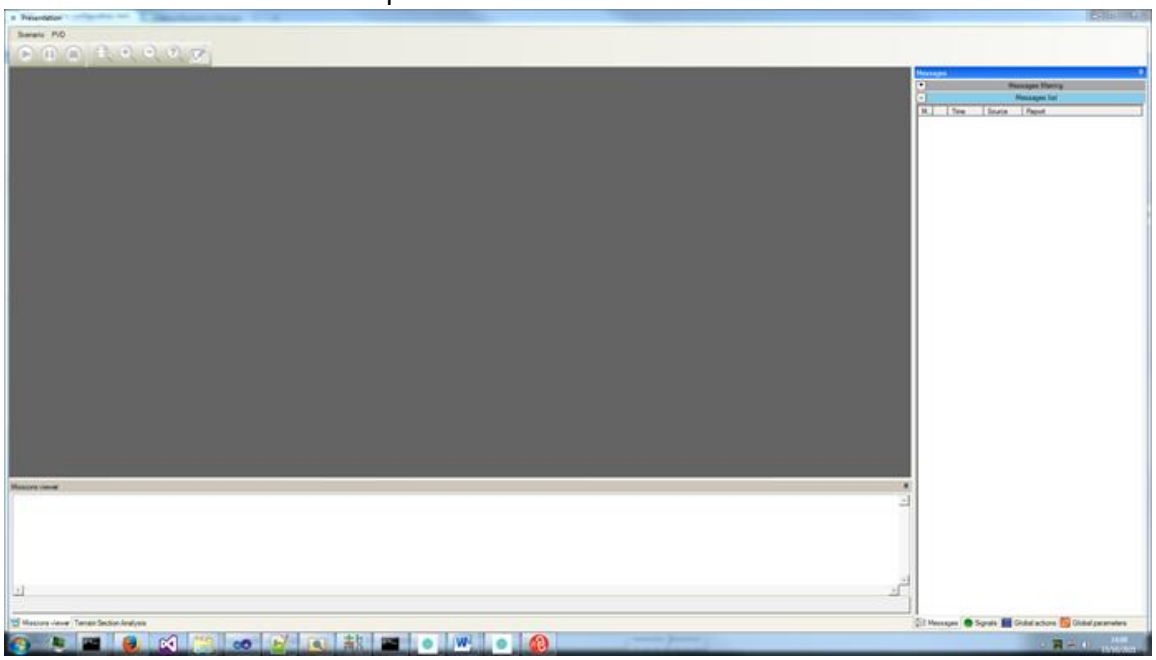



4. Create a preparation session named "PREP".
   Choose the "IOSM" resource Two types of resources are reserved: "PREP IOSC" and "Presentation".
   The session is launched. The following applications are launched:

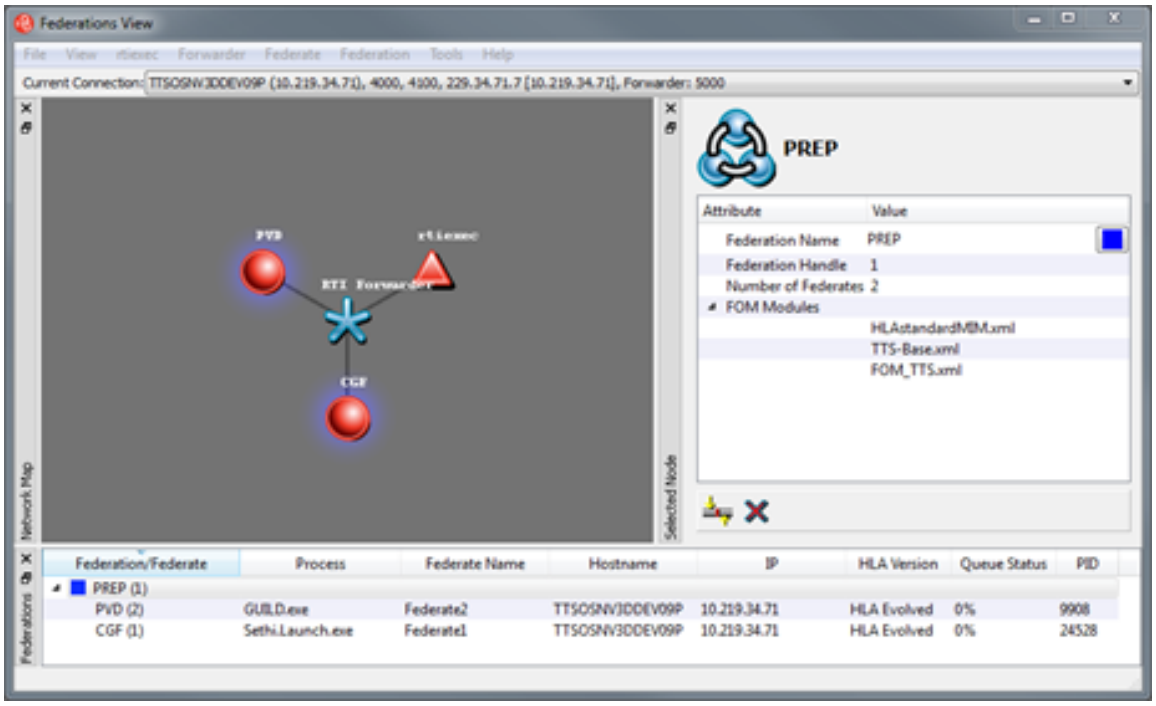



   The "PREP" profile GUI is loaded.

The result is as shown in the picture below:

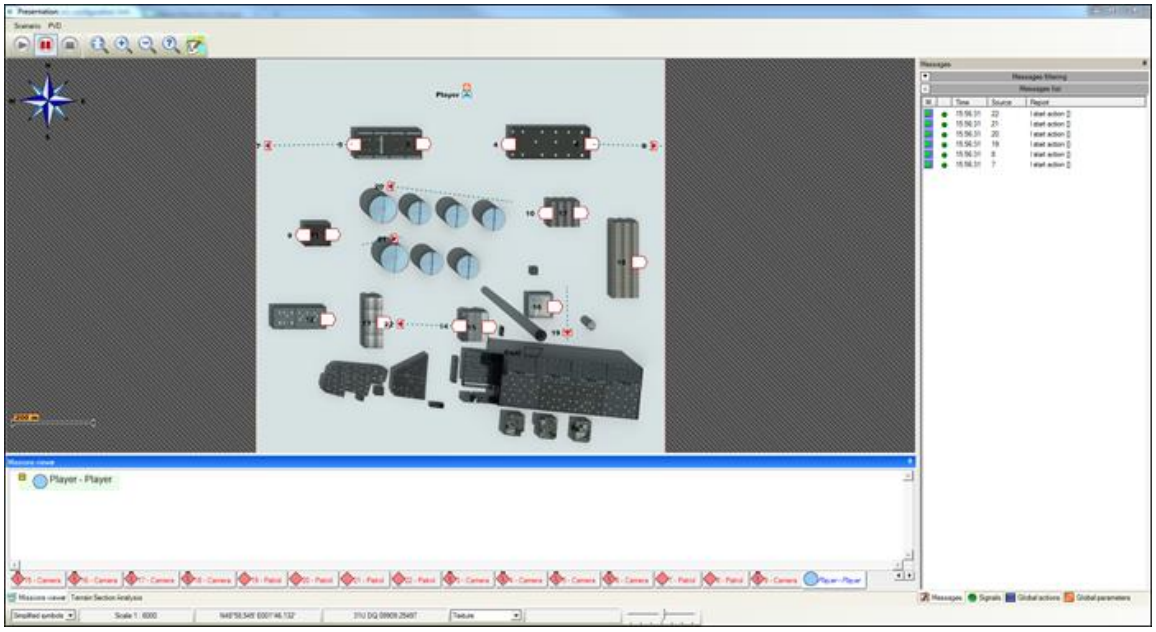


The HLA federates are as follows:

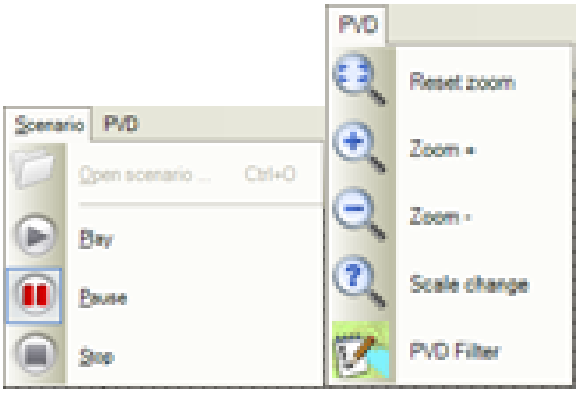



5. Load the scenario: "iv4XR_3.scen".

6. Do "Run" then "Freeze".
   The result is as shown in the image below:

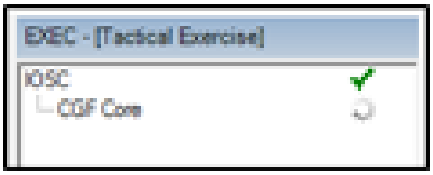


7. The menus are as shown in the images below:



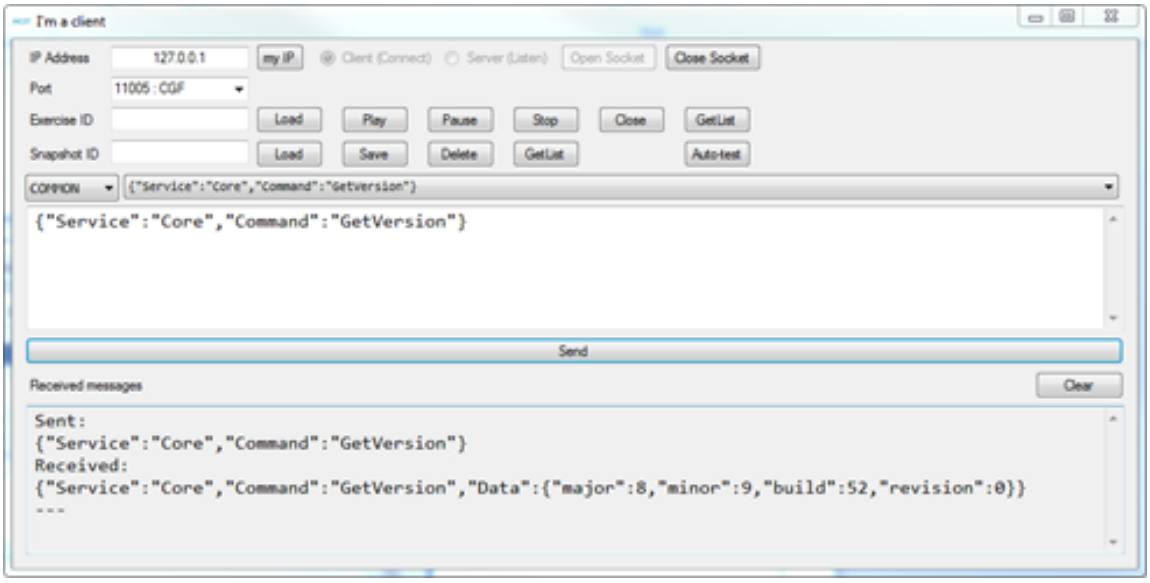

8. Via the CONF HMI, select the "EXEC" session and click the "Delete" button. All applications will stop.

9. Create an execution session named "EXEC".
   Select the "IOSC" resource. A resource type is reserved: "Learning".
   The session starts. The following applications are launched:

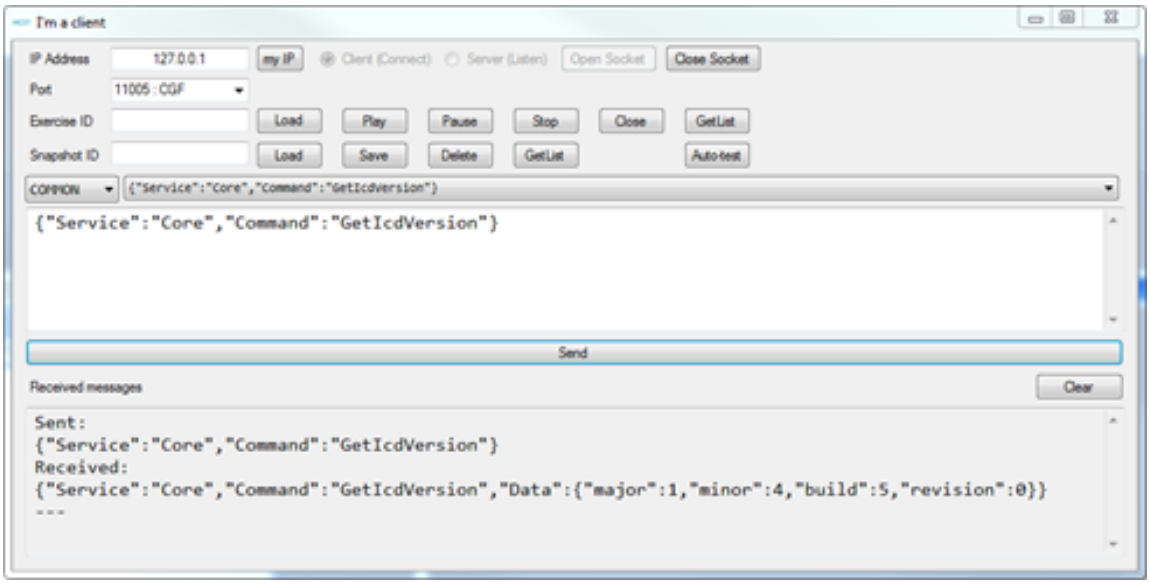



   No HLA or GUI federation is started: only the core CGF is used.

10. Select in the "Port" list the choice "11005: CGF" then click on the "Open Socket" button. Select "COMMON" in the list on the left, then on the "GetVersion" command and click on "Send".

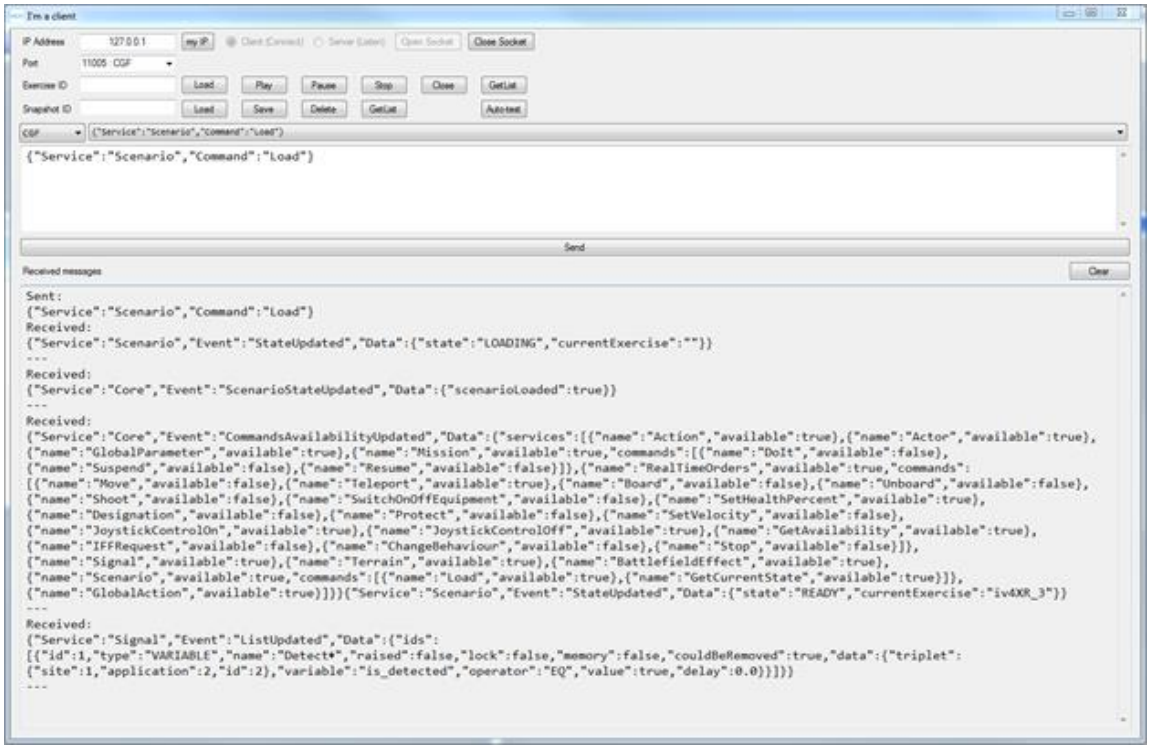



Continue to select "COMMON" in the list on the left, then on the "GetIcdVersion" command and click on "Send".



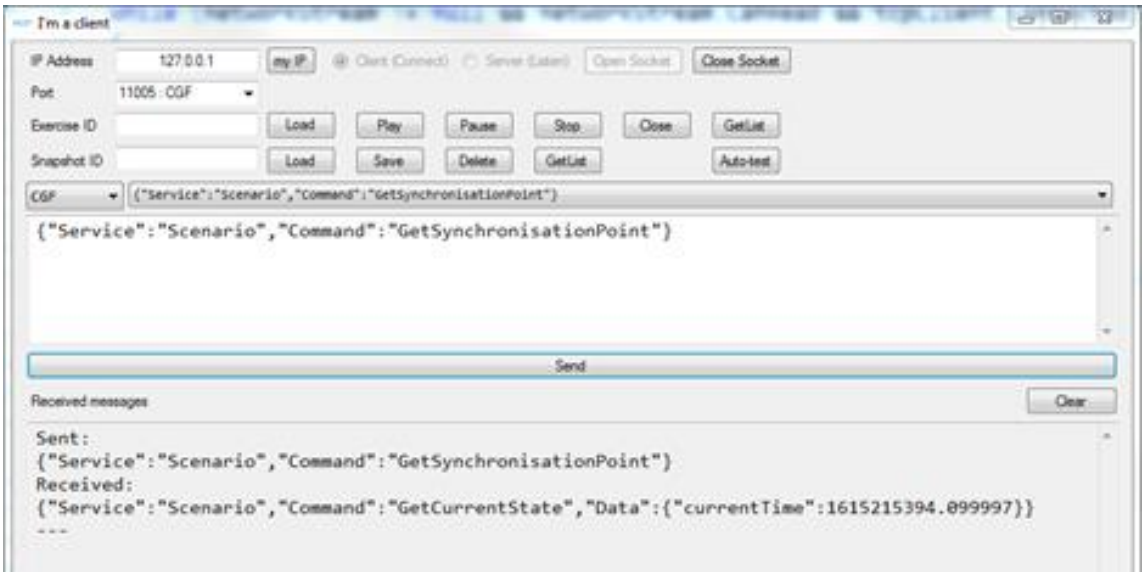

Note: the HCI_Tester is not delivered with the TF-IV4XR, it must be retrieved from the reference TF-ENVEX.

11. Load the scenario "iv4XR_3.scen" via HCI_Tester :

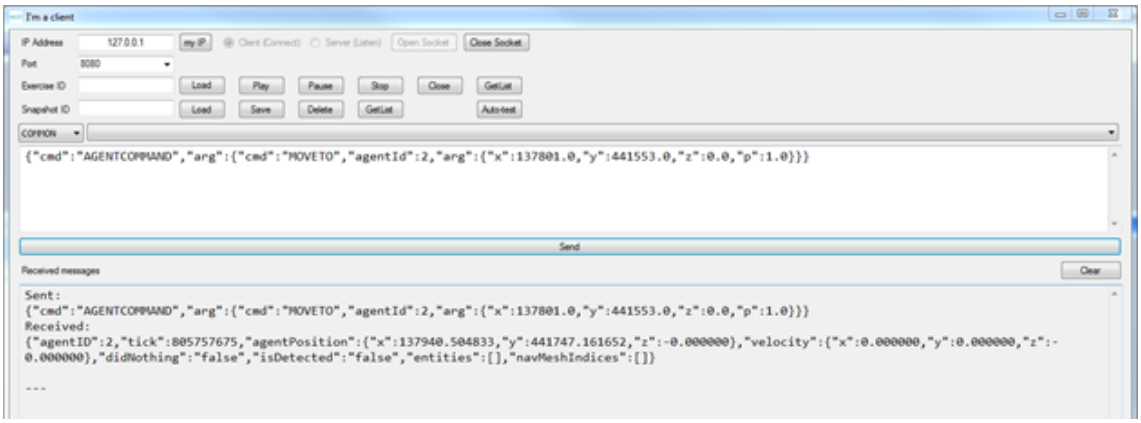



12. Press "Play":

13. Use the "GetSynchronizationPoint" command:

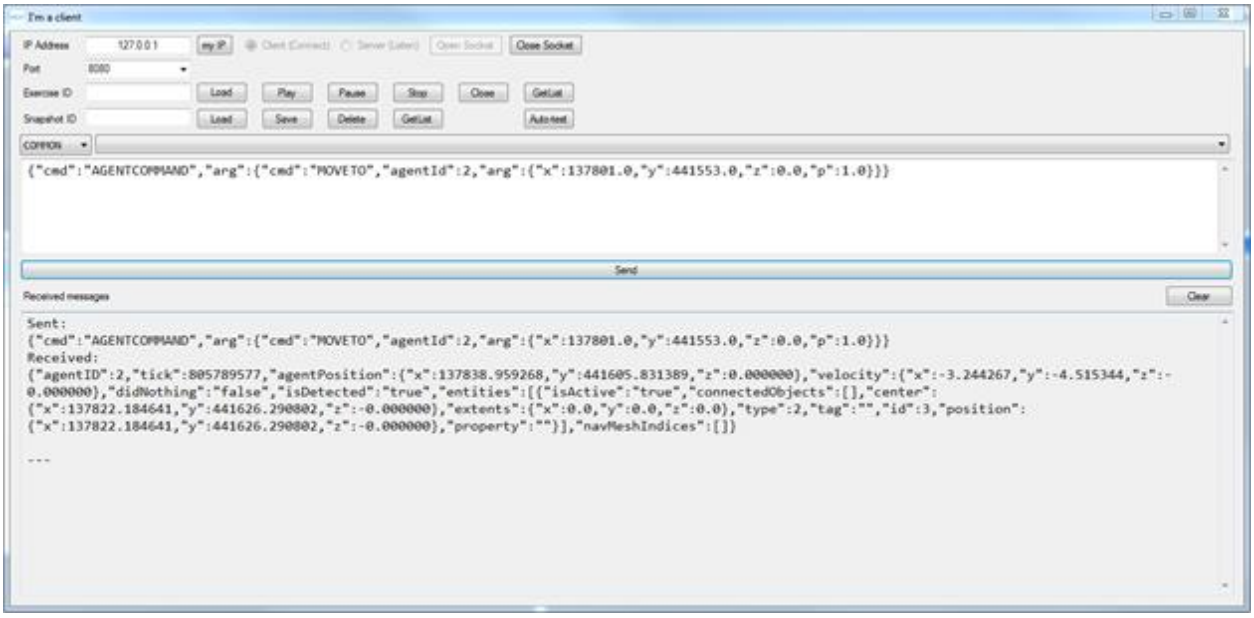



14. Connect to AIEngine via the DDBSocket using port 8080 and send a move command:




*Command used*:
*{"cmd":"AGENTCOMMAND","arg":{"cmd":"MOVETO","agentId":2,"arg":{"x":137801.0,"y":441553.0,"z":0.0,"p":1.0}}}*

15. Wait a few seconds and return the same move order.
    The entity has moved and is now detected (""isDetected": "true"").
    It also detects entities ("entities" is not empty).




16. Via the CONF GUI, select the "EXEC" session and click on the "Delete" button.

    All applications stop.