

HORIZON2020

Deliverable D1.7
Internal check tests and IM2D documentation



D1.7

Internal check tests and IM2D documentation

Matthias Büschelberger, Kathrin Frei, Matteo Bertocchi, Claudio Rosati, and
Arrigo Calzolari



Document information

| | |
|----------------------------------|--|
| Project acronym: | INTERSECT |
| Project full title: | Interoperable Material-to-Device simulation box for disruptive electronics |
| Research Action Project type: | Accelerating the uptake of materials modelling software (IA) |
| EC Grant agreement no.: | 814487 |
| Project starting / end date: | 1 st January 2019 (M1) / 30 th April 2022 (M40) |
| Website: | www.intersect-project.eu |
| Final version: | 29/04/2022 |
| Deliverable No.: | D1.7 |
| Responsible participant: | Fraunhofer (participant number 5) |
| Contributing Consortium members: | AMAT, CNR |
| Due date of deliverable: | 30/04/2022 |
| Actual submission date: | 29/04/2022 |
| Dissemination level: | PU - Public |

Authors: Matthias Büschelberger, Kathrin Frei, Matteo Bertocchi, Claudio Rosati & Arrigo Calzolari.

To be cited as: M. Büschelberger, K. Frei, M. Bertocchi, C. Rosati, and A. Calzolari (2022): Internal check tests and IM2D documentation (final version as of 29/04/2022). EC grant agreement no: 814487, Fraunhofer Gesellschaft Zur Foerderung Der Angewandten Forschung E.V., Freiburg, DE.

Disclaimer:

This document's contents are not intended to replace consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.



Versioning and Contribution History

| Version | Date | Modified by | Modification reason |
|---------|------------|---------------------------|---------------------|
| v.01 | 21/04/2022 | Matthias Büschelberger | First version |
| v.02 | 29/04/2022 | Arrigo Calzolari | Final version |



Contents

| | |
|--|----|
| 1. Executive Summary | 5 |
| 2. Description of the work done | 5 |
| 3. Deviation from planned work in the DoA | 6 |
| 4. Central repository, versioning system and development organisation of IM2D code | 6 |
| 4.1 Main repository and management of dependencies | 6 |
| 4.2 Dockerfiles | 8 |
| 4.3 Docker-registry, versioning and organisation of development | 8 |
| 5. Gitlab CI | 11 |
| 6. Unit tests | 12 |
| 6.1 Basic tests | 13 |
| 6.2 Advanced tests | 13 |
| 6.3 Advanced tests with Hubbard-U parameters | 14 |
| 7. IM2D documentation | 15 |
| 7.1 GUI manual | 15 |
| 7.2 ReadTheDocs for IM2D backend | 16 |
| 7.2.1 Installation guide | 18 |
| 7.2.2 Description of properties, functionalities and ontologies | 19 |
| 7.2.3 User tutorials | 22 |
| 7.3 Swagger UI | 23 |
| Conclusion and outlook | 25 |
| References and links | 26 |
| Acronyms ¹ | 26 |

¹ Acronyms are marked in purple in the text and defined at the end of the document.



1. Executive Summary

The content of this deliverable is related to Task T1.5 (Software engineering up to TRL7) and describes in detail the current status of the **IM2D** code repositories and their dynamic links to each other (Section 4), the implementation of the git-pipeline (Section 5), the unit tests created to check the code functionality and stability (Section 6), and the IM2D documentation on the **REST-API** and app architecture of the simulation box (Section 7). All the actions reported in this deliverable define the quality of the development organisation, the reproducibility and the maintainability of the IM2D codebase. Since there are no considerable deviations from the planned work, the activities converge with common standards for quality assurance measures in software development and, hence, build a solid fundament for future extensions and improvements of the IM2D app without breaking the stability of previous versions.

2. Description of the work done

During the period M25-M40, Fraunhofer IWM has continuously worked on improvements of the IM2D code on the Fraunhofer Gitlab as well as on automated tests for checking the code stability, the code documentation for installation, code capabilities and code usage tutorials. The latter was demonstrated during the Online TechCafé on March 29th 2022 (see D4.9), in collaboration with EPFL and CNR colleagues.

All available versions of the IM2D code developed during the project have been virtualized through Docker and are available as light-weighted virtual machines (containers) in the Docker registry of the Fraunhofer Gitlab. This allows any registered user to easily download the related containers on any individual local machine and run the application without installing and configuring the code components from scratch. The advantages of this technology for the IM2D framework were previously discussed in D1.6² (GUI deployment).

The actions described above boost the reproducibility, code quality, stability and maintainability of the whole application.

According to the DoA, the T1.5 activities include:

- a) Creation of a central code repository and versioning system
- b) Organisation of software development
- c) Definition and implementation of a testing protocol for each of the components of the IM2D box, and of the interfaces, wrappers and plugins for interoperability
- d) Construction of a battery of tests to feed the protocol defined above
- e) Production of software documentation, including description of models, physical equations and material relations; code architecture, installation and user guide, and test execution examples.

² <https://intersect-project.eu/wp-content/uploads/2022/04/D1.6.pdf>
www.intersect-project.eu



In the course of this document, Section 4 (*Central repository, versioning system and development organization of IM2D*) addresses the points (a) and (b); Section 5 (*Gitlab CI*) discusses action (c); Section 6 (*Unit tests*) relates to activity (d); point (e) is described in Section 7 (*IM2D documentation*).

3. Deviation from planned work in the DoA

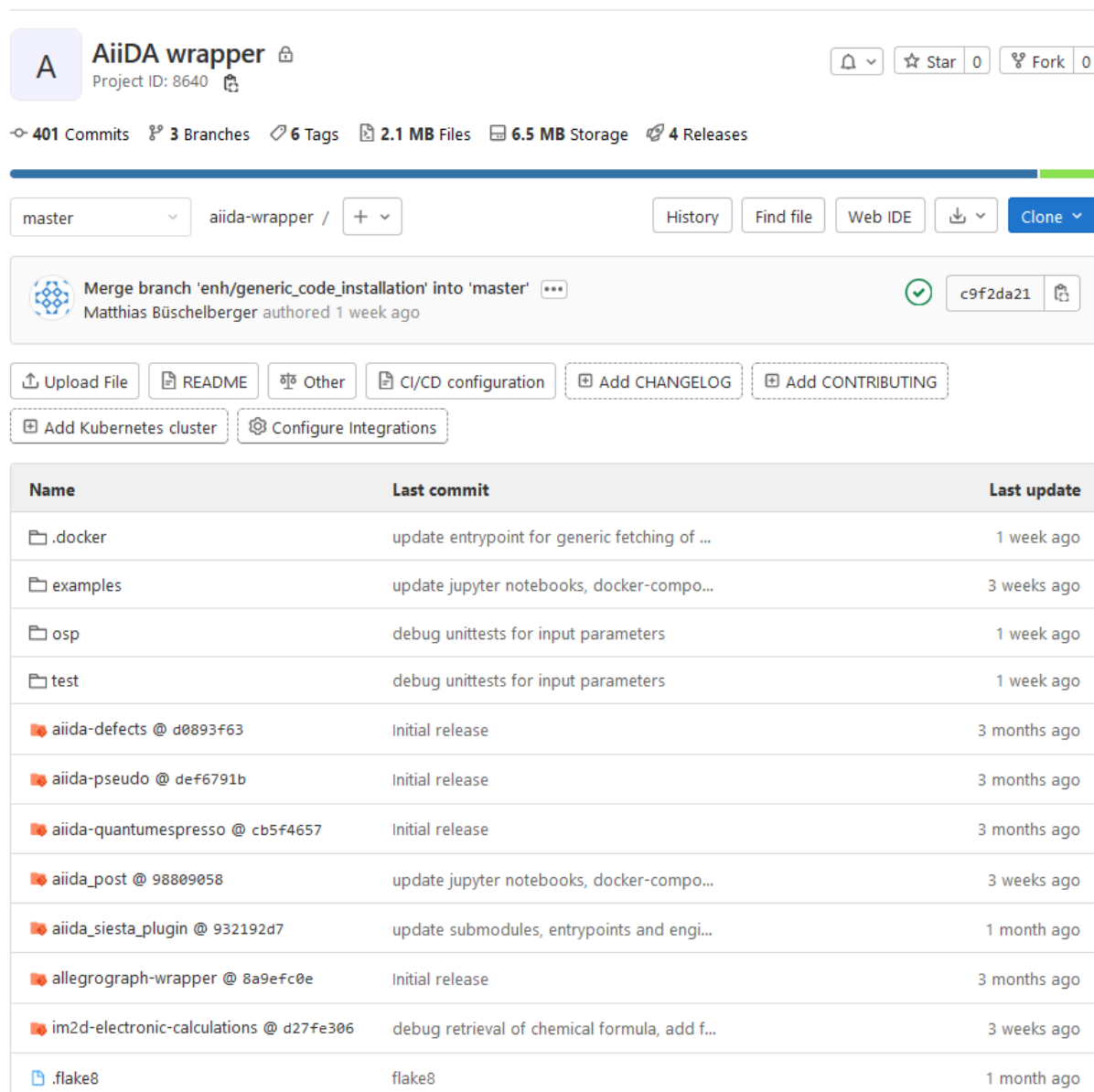
Although there are no considerable deviations from the planned actions, we report the following aspects:



- The DoA proposed to include the detailed descriptions of the models, equations and material relations which are used in the IM2D framework. However, since a large collection of the workflows facilitated in AiiDA through the related plugins - such as, e.g., AiiDA-Siesta and AiiDA-Quantum-Espresso - were developed independently of IM2D, detailed descriptions of the underlying models and equations are not included in the developed documentation. On the contrary, specific code documentation, publications and INTERSECT-deliverables are linked and cited as needed. This also applies to the descriptions pertaining to the models of SESTA and Quantum-Espresso (QE) codes.
- Additionally, the consortium partners are discussing the publication of the app components on Github. Therefore, the development system might not be continued on the private Fraunhofer Gitlab but displaced on a public repository on Github.

4. Central repository, versioning system and development organisation of IM2D code

4.1 Main repository and management of dependencies



At the time of submission of this deliverable, the main repository [1] includes all of the IM2D-app components and is hosted on the Fraunhofer Gitlab. This repository, called *aiida-wrapper*, contains the semantic interface of SimPhoNy and AiiDA as well as the SimPhoNy-REST API that uses the wrapper in order to communicate with any HTTP-client, such as the IM2D GUI developed in the project (see D1.6). A screenshot of the repository with its contents is presented in Figure 1.



AiiDA wrapper  Project ID: 8640 

🔔 401 Commits 3 Branches 6 Tags 2.1 MB Files 6.5 MB Storage 4 Releases

master aiiida-wrapper / + History Find file Web IDE Clone

Merge branch 'enh/generic_code_installation' into 'master'  c9f2da21 
Matthias Büschelberger authored 1 week ago

Upload File README Other CI/CD configuration Add CHANGELOG Add CONTRIBUTING

Add Kubernetes cluster Configure Integrations

| Name | Last commit | Last update |
|---|---|--------------|
| .docker | update entrypoint for generic fetching of ... | 1 week ago |
| examples | update jupyter notebooks, docker-compo... | 3 weeks ago |
| osp | debug unittests for input parameters | 1 week ago |
| test | debug unittests for input parameters | 1 week ago |
| aiida-defects @ d0893f63 | Initial release | 3 months ago |
| aiida-pseudo @ def6791b | Initial release | 3 months ago |
| aiida-quantumespresso @ cb5f4657 | Initial release | 3 months ago |
| aiida_post @ 98809058 | update jupyter notebooks, docker-compo... | 3 weeks ago |
| aiida_siesta_plugin @ 932192d7 | update submodules, entrypoints and engi... | 1 month ago |
| allegrograph-wrapper @ 8a9efc0e | Initial release | 3 months ago |
| im2d-electronic-calculations @ d27fe306 | debug retrieval of chemical formula, add f... | 3 weeks ago |
| .flake8 | flake8 | 1 month ago |

Figure 1: Screenshot of the *aiida-wrapper* repository on the Fraunhofer Gitlab. The repository hosts the source code for the semantic interfaces and the operating REST-API of SimPhoNy (“osp”-directory), the dynamically linked dependencies (submodules: “aiida-defects”, “aiida-quantumespresso”, “aiida-pseudo”, etc), the implementations of the unit tests for code stability (“tests”-directory) and the related Docker files- and configurations (“.docker”-directory).

The *aiida-wrapper* repository is dynamically linked (via git-submodules) to other git-repositories such as the AiiDA-plugins of *aiida-quantumespresso* [2], *aiida-siesta* [3], *aiida-defects* [4], *aiida-pseudo* [5], and *aiida-post* [6]. Other repositories, for example the *allegrograph-wrapper* [7] (a SimPhoNy-interface to the triplestore-software of Allegrograph), as well as the IM2D-ontology [8] based on EMMO (called *im2d-electronic-calculations*), are included as submodules as well. Since most of the mentioned plugins are hosted on public Github (exceptions: *allegrograph-wrapper*, *aiida-post-plugin*, *im2d-electronic-calculations-ontology*), this dynamic linking of the dependencies assures a reproducibility and an easy



installation procedure of the whole IM2D-code from scratch, without statically including the dependency source-code in the *aiida-wrapper*-repository. The linking functionality to submodules is also guaranteed, in the case the repository is moved to another Git-host. Additionally, submodules are pinned to certain versions/commits of the remote repository, so that previous versions of the IM2D-code can be recovered, even if the dependency is updated.

Currently, the *aiida-defects*-plugin is privately available on Github; while *allegrograph-wrapper*, *aiida-post*-plugin, and *im2d-electronic-calculations-ontology* are hosted on the private Fraunhofer Gitlab. At the time of this report, a move of these repositories to public Github is under discussion by the consortium.

4.2 Dockerfiles

Submodules are especially useful for building Docker-images, which provide a fast and easy deployment on any arbitrary host-machine, such as production HPC servers or local systems. The Docker and the Docker-compose files for the deployment of the IM2D app are included in the *aiida-wrapper*-repository. The images for the virtual machines (called *containers* in Docker) can be pushed and pulled from the Docker-registry of the Fraunhofer Gitlab. This allows the user to access the images without building them from scratch: A potential user of the IM2D box only needs to install Docker on the operating system, clone the *aiida-wrapper* repository and pull the containers from the Docker-registry. Building the images from scratch should only be considered when a new release of the app is published.

The installation guide is reported below in the IM2D documentation (see chapter 7 – *IM2D documentation*). For further details about the deployment through Docker, see D1.6.

4.3 Docker-registry, versioning and organisation of development

The development of the application exploited the standard capabilities of the git-software-development tool suite. This allows for parallel development on different branches of the repository, without breaking stable versions of previous releases. Additionally, each release features notes that describe the newest developments and the bugs fixed, with respect to previous versions.

The commit-history, as shown in Figure 2, makes the changes in the code, among different variants, recoverable and traceable. The Docker-images (Figure 3) of each release are continuously pushed to the Docker-registry (see Figure 4) of the Fraunhofer-Gitlab.

Deliverable D1.7

Internal check tests and IM2D documentation

SimPhoNy > Wrappers > AiIDA wrapper > Tags

Tags give the ability to mark specific points in history as being important

Filter by tag name Updated date

v0.5 ☒

→ c9f2da21 · Merge branch 'enh/generic_code_installation' into 'master' · 1 week ago

🔗 Release v0.5

* add generic installation for new codes and remote computers in AiIDA

v0.4.1 ☒

→ 12686f16 · Merge branch 'dev' into 'master' · 2 weeks ago

🔗 Release v0.4.1

Hotfix: pin dependencies of SimPhoNy

v0.4 ☒

→ d335abf1 · Merge branch 'dev' into 'master' · 2 weeks ago

🔗 Release v0.4

Changes in V0.4:

- * separate QE from the AiIDA-Container
- * Add Siesta-container
- * Add Siesta-plugin
- * enable ssh-connections to QE/Siesta
- * add first unittests
- * bring back the git-pipeline
- * make a separate property-mapping for AiIDA-post
- * add Jupyter-notebooks with tutorials
- * add healthchecks for docker containers
- * add parser for syntactic inputs (e.g. JSON) in order to accelerate the submission

v0.3

→ d12bb8c4 · Merge branch 'dev' into 'master' · 2 months ago

🔗 Release v0.3

v0.2

→ 5159b1c1 · Merge branch 'dev' into 'master' · 3 months ago

Introduce json-ld endpoints, debug some get-methods, while waiting for response of post-method, add docker secrets.

v0.1

→ 7c163e9f · Merge branch 'dev' into 'master' · 3 months ago

Initial release for MarketPlace

Figure 2: Release history with release notes of the aiida-wrapper plugin for SimPhoNy.

Bugs, feature requests or unexpected behaviours can be reported through the common usage of issues on git and, therefore, can be related to ongoing developments and new releases.

The remaining functionalities of git that were not touched in this section are described in the previous deliverable (D1.2³ - *Creation of code repository and versioning system*).

³ <https://intersect-project.eu/wp-content/uploads/2022/04/D1.2.pdf>
www.intersect-project.eu



Root image 🔗

5 tags 🔗 Cleanup disabled 🔄 Last updated 3 months ago

Name ▾
↓

☐ 5 tags

☐ **v0.3** 🔗 ⋮

Published 2 months ago

Digest: 52e0a24

413.80 MiB

🕒 Published to the *simphony/wrappers/aiida-wrapper* image repository at 07:39 GMT+0100 on 2022-02-10

📄 Manifest digest: sha256:52e0a24bedf71c112af028fa0399d677ab39bd7eea76c310b4dbc24fea2cbc96 🔗

🔗 Configuration digest: sha256:918c1702b5affcaea19d88671f7cbdfef5fc619f1cb519243688548418d1a9067 🔗

☐ **v0.4** 🔗 ⋮

Published 2 weeks ago

Digest: 0885161

414.29 MiB

🕒 Published to the *simphony/wrappers/aiida-wrapper* image repository at 04:38 GMT+0200 on 2022-04-04

📄 Manifest digest: sha256:088516134669490afdab9b89df4f92ead8561d1edd6d0501c32c84c5cbf48adf 🔗

🔗 Configuration digest: sha256:6412df348e64fdc15dc64f76fcc6ce596e89b8063718e235a1dc1d516cc380d0 🔗

☐ **v0.4.1** 🔗 ⋮

Published 2 weeks ago

Digest: 0885161

414.29 MiB

🕒 Published to the *simphony/wrappers/aiida-wrapper* image repository at 04:38 GMT+0200 on 2022-04-04

📄 Manifest digest: sha256:088516134669490afdab9b89df4f92ead8561d1edd6d0501c32c84c5cbf48adf 🔗

🔗 Configuration digest: sha256:6412df348e64fdc15dc64f76fcc6ce596e89b8063718e235a1dc1d516cc380d0 🔗

☐ **v0.5** 🔗 ⋮

Published 1 week ago

Digest: 74482c0

414.30 MiB

🕒 Published to the *simphony/wrappers/aiida-wrapper* image repository at 01:19 GMT+0200 on 2022-04-11

📄 Manifest digest: sha256:74482c0e2810d1945a34e628d08811b1c85706cca999cf60631df9f81a18a7a2 🔗

🔗 Configuration digest: sha256:0a26100de20e4b2d5d33f9978fe422c6f7950f8a53196340d999ec78d9bef6d4 🔗

Figure 3: Overview of Docker-images of each IM2D-release on the Docker-registry of the Fraunhofer Gitlab. The AiiDA-, Quantum-Espresso-, and Siesta-images are also hosted on this platform but are not displayed in this figure.

Deliverable D1.7

Internal check tests and IM2D documentation































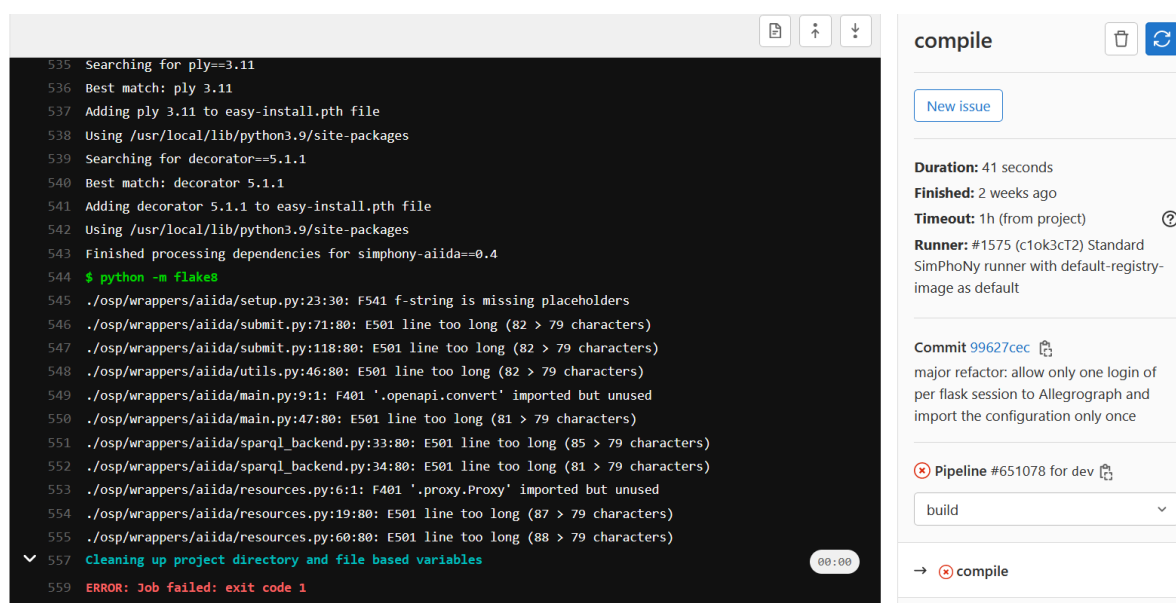
| All 211 | Pending 0 | Running 0 | Finished 211 | | | |
|-------------------|-----------|---|--|-------|--|---|
| Status | Name | Job | Pipeline | Stage | Duration | Coverage |
| <div>passed</div> | compile | #1751941  hubbard  d0a26770 | #663456 by  | build |  00:00:23  20 hours ago |  |
| <div>passed</div> | compile | #1751913  hubbard  f23b0b63 | #663438 by  | build |  00:00:23  21 hours ago |  |
| <div>passed</div> | compile | #1751826  hubbard  c7cc4106 | #663410 by  | build |  00:00:23  21 hours ago |  |
| <div>passed</div> | compile | #1751800  hubbard  d4f872b8 | #663399 by  | build |  00:00:36  22 hours ago |  |
| <div>passed</div> | compile | #1751686  hubbard  266445b7 | #663335 by  | build |  00:00:24  23 hours ago |  |

Figure 4: List of the most recent pipelines for the aiida-wrapper repository.

5. Gitlab CI

The continuous integration (CI) on Gitlab is a standardised process that can be fully managed by a markup file written in yaml-format. This specification triggers a pipeline of Docker-containers, which execute a suite of tests every time a commit has been pushed to the *aiida-wrapper*-repository. Each pipeline process and its attributes (e.g., commit, duration, etc.), are stored and available on the respective Gitlab repository, as shown in Figure 5. This assures a protocol for tracing changes in the codebase, whether new features run as expected, and whether all functionalities of the application still work or whether the new changes broke pre-existing capabilities.



```

535 Searching for ply==3.11
536 Best match: ply 3.11
537 Adding ply 3.11 to easy-install.pth file
538 Using /usr/local/lib/python3.9/site-packages
539 Searching for decorator==5.1.1
540 Best match: decorator 5.1.1
541 Adding decorator 5.1.1 to easy-install.pth file
542 Using /usr/local/lib/python3.9/site-packages
543 Finished processing dependencies for simphony-aiida==0.4
544 $ python -m flake8
545 ./osp/wrappers/aiida/setup.py:23:30: F501 f-string is missing placeholders
546 ./osp/wrappers/aiida/submit.py:71:80: E501 line too long (82 > 79 characters)
547 ./osp/wrappers/aiida/submit.py:118:80: E501 line too long (82 > 79 characters)
548 ./osp/wrappers/aiida/submit.py:46:80: E501 line too long (82 > 79 characters)
549 ./osp/wrappers/aiida/main.py:9:1: F401 '.openapi.convert' imported but unused
550 ./osp/wrappers/aiida/main.py:47:80: E501 line too long (81 > 79 characters)
551 ./osp/wrappers/aiida/sparql_backend.py:33:80: E501 line too long (85 > 79 characters)
552 ./osp/wrappers/aiida/sparql_backend.py:34:80: E501 line too long (81 > 79 characters)
553 ./osp/wrappers/aiida/resources.py:6:1: F401 '.proxy.Proxy' imported but unused
554 ./osp/wrappers/aiida/resources.py:19:80: E501 line too long (87 > 79 characters)
555 ./osp/wrappers/aiida/resources.py:60:80: E501 line too long (88 > 79 characters)
557 Cleaning up project directory and file based variables
559 ERROR: Job failed: exit code 1

```

compile

New issue

Duration: 41 seconds
Finished: 2 weeks ago
Timeout: 1h (from project)
Runner: #1575 (c1ok3CT2) Standard SimPhoNy runner with default-registry-image as default

Commit 99627cec
major refactor: allow only one login of per flask session to Allegrograph and import the configuration only once

Pipeline #651078 for dev
build

→ **compile**

Figure 5: Example of a failed pipeline for a certain commit due to bad code style.



The Docker-images for the AllegroGraph-software are pulled from the public Dockerhub. The same also holds for the postgres-database used by AiiDA. The Docker containers hosting AiiDA, QE and Siesta are run from the pre-existing images from the Fraunhofer Gitlab Docker-registry. On the other hand, the SimPhoNy-container (which only comprises the source code of the *aiida-wrapper*-repository) is entirely rebuilt for testing the implementations within the pipeline. The rebuilding includes the installation of OSP-core and the *allegrograph-wrapper* as well as the installation of the *im2d-electronic-calculations-ontology* in OSP-core and the setup of the databases on AllegroGraph.

After rebuilding is completed, a set of checks for the code style and a variety of unit tests for the simulation box are run within the SimPhoNy-container. In case of a test failure, the underlying protocol of the Gitlab pipeline raises an error and also forwards the traceback of this exception towards the code-maintainer. Since the pipeline can be automatically repeated in fixed intervals without pushing any new commits, this functionality represents the most suitable technology for assuring the long-time stability of the IM2D app and all of its components. The tests are further explained in Section 6 (*Unit tests*).

If the repositories will be moved to public Github, the protocol has to be translated into the format of the new platform, because the yaml-specifications between Gitlab and Github are not mutually compatible.

6. Unit tests

Unit tests are standard procedures in software development in order to provide a self-controlled mechanism to verify that the developed functionalities of the codebase return the data expected in specific data types, serialisation formats and accuracies. They react very sensitively to any small change or error in the code and detect even very slight differences in the data returned by the software. Since unit tests can be executed automatically and repeated infinitely, they represent a key technology for code stability and are easily applicable for CI/CD (see Section 5 - *Gitlab CI*).

Due to the complexity of the overall IM2D simulation box, the unit tests were categorised into *basic*, which succeed after a short period of time, and *advanced*, which can take up to one hour since they run several workflows in AiiDA, trigger simulations in QE and SIESTA for a standard material, and compare the output of the executed simulations with expected values. All of these tests run fully automatically and independently from the action of the software developer.

Additionally, separate tests were implemented for applying the Hubbard-U parameters of QE to the AiiDA-workflows. Since these parameters were added very recently to the advanced



user profile (*persona* D1.1⁴) of the simulation box, at present they are separated from the common advanced tests, which deliver more stable results. Finally, they will be integrated into the entire suite of tests.

6.1 Basic tests

The translation between the syntactic and semantic data is one of the main functionalities of the *aiida-wrapper* and represents one potential source of error for a failed workflow. The basic unit tests for the *aiida-wrapper* repository check the conversion of semantic CUDS (ontology individuals representing materials entities and simulation in/output specifications – see D1.3⁵ and D2.5⁶) into the syntactic JSON data to be submitted to the AiiDA-REST API. In order to make the test more streamlined, the simulation is mocked and dummy data is returned, so that the back-and-forth conversion of CUDS can be tested efficiently.

Additionally, the forwarding of the information about the high-level requirements for each persona (ergo: “which are the inputs for a certain workflow?” for each user profile) through the REST-API of SimPhoNy is tested. The SPARQL-queries retrieve information through the *allegrograph-wrapper* from the ontology-classes deployed at the AllegroGraph triplestore. The result holds input information such as the related ontology-class-IRI of a physical quantity, its atomic units, its physical dimensions, further-reading-suggestions and its expected data-types (see D2.5). This functionality needs to be tested continuously when the ontology is maintained since it is crucial information to be parsed by the GUI. Small changes in the formatting returned by the REST-service may cause erroneous displaying in the GUI.

Furthermore, the functionality of forwarding recommendations for different simulation accuracies for these input parameters is tested for a standard material (silicon). This type of unit test is essential because the response coming from the REST-API must be correctly processed by the GUI and the implementation is a complex mix of a SPARQL-query towards the AllegroGraph triplestore for the given persona-input and an http-query for a QE-protocol to the AiiDA-container.

6.2 Advanced tests

As mentioned above, advanced unit tests trigger workflows in AiiDA for the computation of material properties (such as the band gap, band structure, dielectric constant, effective mass, etc.) by running simulations in QE and SIESTA. Each computation runs on low accuracy and a standard material (silicon) for roughly 15 minutes. A tolerated deviation for the returned values, e.g., of the band gap calculated by AiiDA, is currently 1e-13 eV. Nevertheless, it has to be stressed that this test is conceived only for assuring software stability, not for validating the feasibility of the returned values.

⁴ <https://intersect-project.eu/wp-content/uploads/2022/04/D1.1.pdf>

⁵ <https://intersect-project.eu/wp-content/uploads/2022/04/D1.3.pdf>

⁶ <https://intersect-project.eu/wp-content/uploads/2022/04/D2.5.pdf>



Advanced unit tests are executed for three different user profiles (basic, intermediate and advanced) and different serialisation formats (JSON for direct syntactic input or JSON-LD for semantic data from CUDS). Thus, a high duration of the whole pipeline process can be reasonably expected. Therefore, we decided to separate these advanced tests from the basic unit tests, so that only basic tests are triggered in the Gitlab pipeline for each commit, but advanced tests are solely executed on merge requests and commits to the master branch. A code snippet of such a unit test is shown in Figure 6.

```

22 class TestBandstructureSyntactic(TestCase):
23
24     PATH = os.path.dirname(os.path.realpath(__file__))
25
26     @mock.patch.object(Proxy, "get", side_effect=MockJsonReponse.get)
27     @mock.patch.object(Proxy, "post", side_effect=MockJsonReponse.post)
28     def test_band_structure_post(self, mock_post_response, mock_get_response):
29         Si = os.path.join(self.PATH, "Si.json")
30
31         bandGap = 0.6048960165406
32         effectiveMass = 1.2643012053893
33         relaxedEnergy = -310.56823314965
34
35         url_nodes = "/api/v4/intersect/nodes/create/structure"
36         iana_url = "https://www.iana.org/assignments/media-types/"
37         url_post = f"/api/v4/intersect/submit"
38         url_status = "/api/v4/intersect/status"
39         url_cuds = "/api/v4/intersect/cuds"
40

```

Figure 6: Code snippet from an advanced unit test for a basic workflow of the band gap, effective mass and relaxed energy of silicon. Variables in the lines 31-33 are values normally returned by AiiDA for a fast computation. The values have not been validated; they are only used as a reference for software stability. This test has been consecutively executed with success in the latest development stages with a tolerated deviation of $1e-13$ for each value.

6.3 Advanced tests with Hubbard-U parameters

In the last part of the project, new parameters, such as the Hubbard-U, were added for the *advanced* persona profile. Since the simulations do not currently return stable results, the corresponding unit tests are separate from those mentioned in Sections 6.1 and 6.2, and are therefore not contained in the Gitlab pipeline at present. The success of the computation depends on the parameterization of the inputs, the pseudo-potentials used by AiiDA, and the version of QE used by AiiDA. Potential materials to test these parameters are WO_3 and ZnO . The workflow for using the Hubbard-U parameters of QE needs to be fine-tuned in order to assure the stability of the simulation results. At present, the band gap-workchains in AiiDA trigger a relaxation by default before a band-structure calculation run is executed in QE. This changes the distribution of the atoms in the unit cell and, consequently, the symmetry of the structure. In order to successfully and reasonably apply the Hubbard-U parameters in the IM2D



simulation box, the workflows have to be modified to make a relaxation optional when the input material is considered to be already relaxed.

7. IM2D documentation

The manuals and documentation for the IM2D simulation toolbox are split into the following components:

- the user manual of the GUI,
- the documentation of the backend services of SimPhoNy and AiiDA,
- the documentation of the REST-API through the Swagger user interface (UI).

The user manual of the GUI is a PDF-document provided by AMAT, which provides a detailed description for end users about the graphical frontend and its main operation instructions (Figure 7). The documentation of the backend services is a ReadTheDocs-page provided by Fraunhofer and focuses on the toolbox capabilities and functionalities. The IM2D ontology and the app architecture are also described in the manual, along with the installation guide and the tutorials for the REST-API of IM2D. The documentation of the REST-API through the Swagger UI, a graphical interface used as an index page for the IM2D REST-API, reports on all available endpoints of the Flask-server used by SimPhoNy.

7.1 GUI manual

The GUI software and the related manual for the IM2D box can be requested by contacting the INTERSECT coordinator (intersect@nano.cnr.it) or the AMAT customer service.

As already stated in D1.6 (GUI deployment), the graphical user interface, which was specifically developed for the IM2D toolbox, is copyrighted by AMAT and hence it is a proprietary software. This leads to a separation between the documentation of the GUI and the backend services of SimPhoNy & AiiDA.

The user guide for the IM2D frontend describes how the user interface is organised, how the different plugins (AiiDA, SimPhoNy, Optimade) can be accessed, and how they can be configured. The manual also contains a detailed description of all the available features and operations.

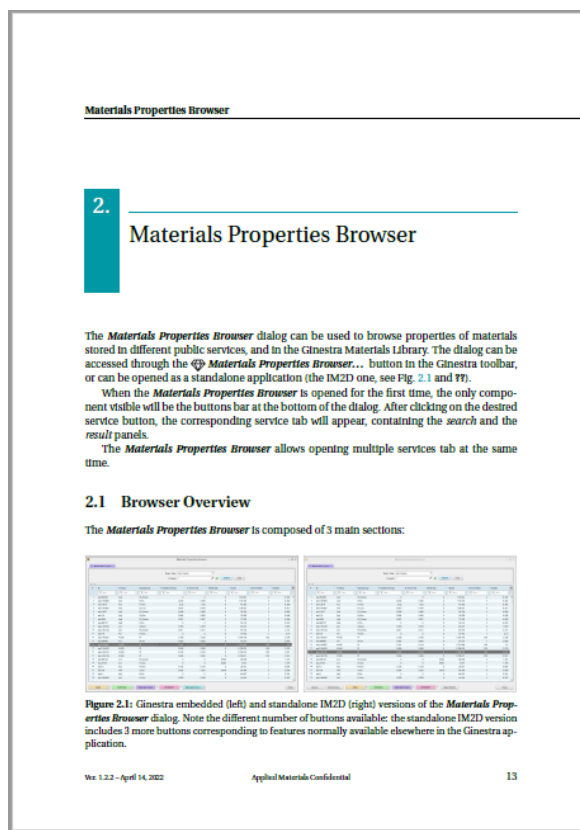
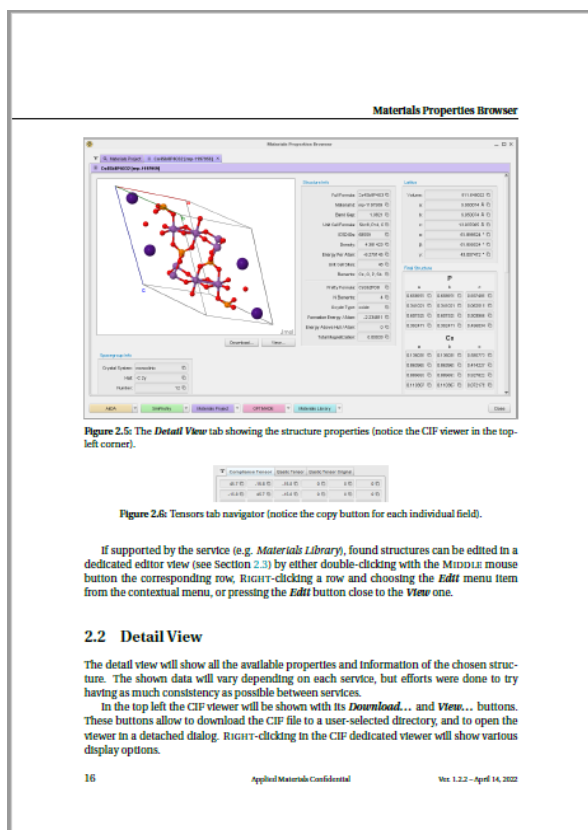


Figure 7: Screenshots of the manual for IM2D graphical user interface copyrighted by AMAT.

7.2 ReadTheDocs for IM2D backend

ReadTheDocs has become one of the most popular documentation tools in OpenSource-software development, since it provides a versioning system compatible with git and therefore it can be easily managed, updated and deployed on the Fraunhofer Gitlab platform. At the moment, the documentation for the IM2D backend services of SimPhoNy, AiiDA, QE and Siesta is hosted under <https://intersect.pages.fraunhofer.de/docs/> (see Figure 8) and managed by the corresponding git repository [9] (see Figure 9). The documents are written in Markdown and hence can very easily be exported as PDF or HTML. Furthermore, the integration of JupyterNotebooks is trivial, so that the tutorials for interacting with the REST-API from the TechCafé (organised on 29th of March 2022) can be included in the documentation on the fly.

At the time of this report, the documentation is only accessible for registered Fraunhofer Gitlab-users. However, the documentation repository can easily be migrated to any other git-host such as Github, which would be the case if the *aiida-wrapper* repository would be moved to public Github as well.

It is worth noticing that the process of writing the documentation is a continuous process. Hence, the content is evolving in time and might be incomplete until new issues or missing information are reported.

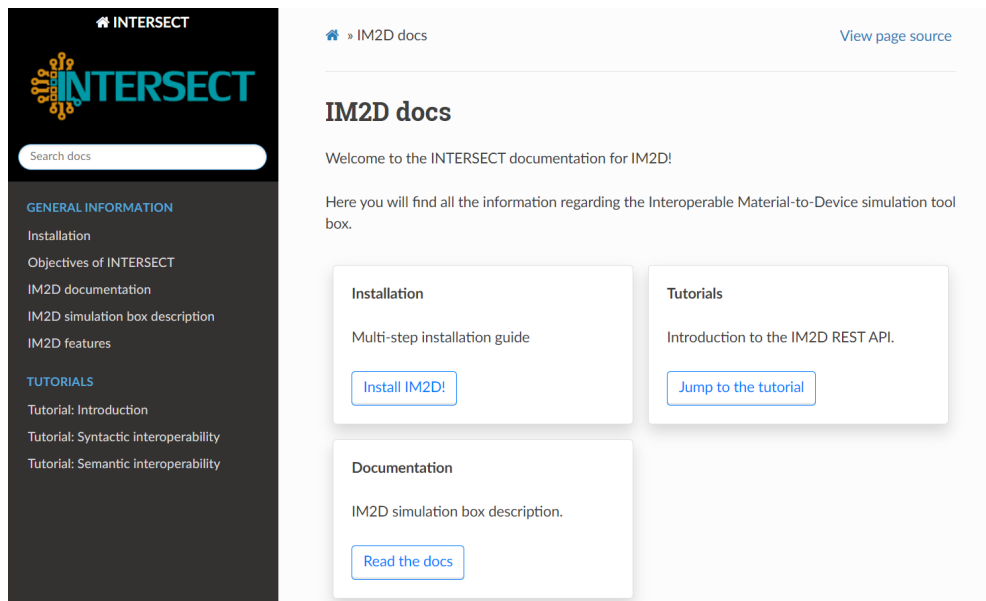


Figure 8: Index page of the IM2D documentation under <https://intersect.pages.fraunhofer.de/docs>.

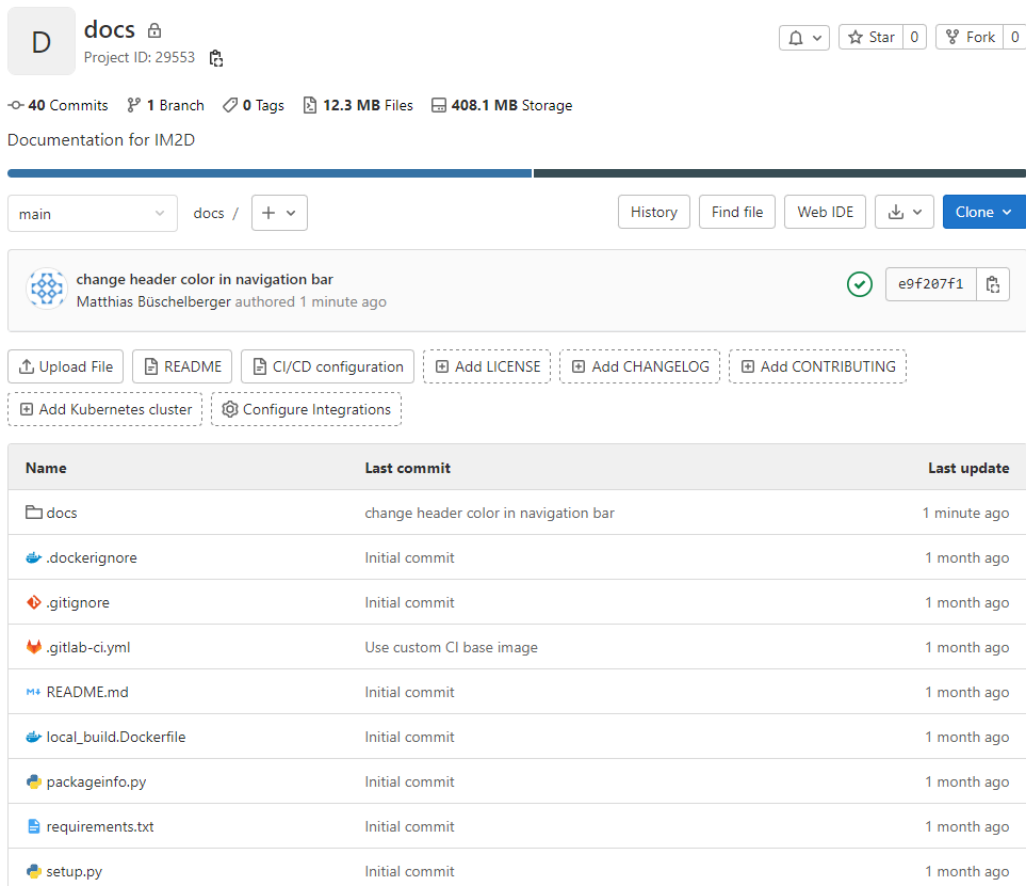
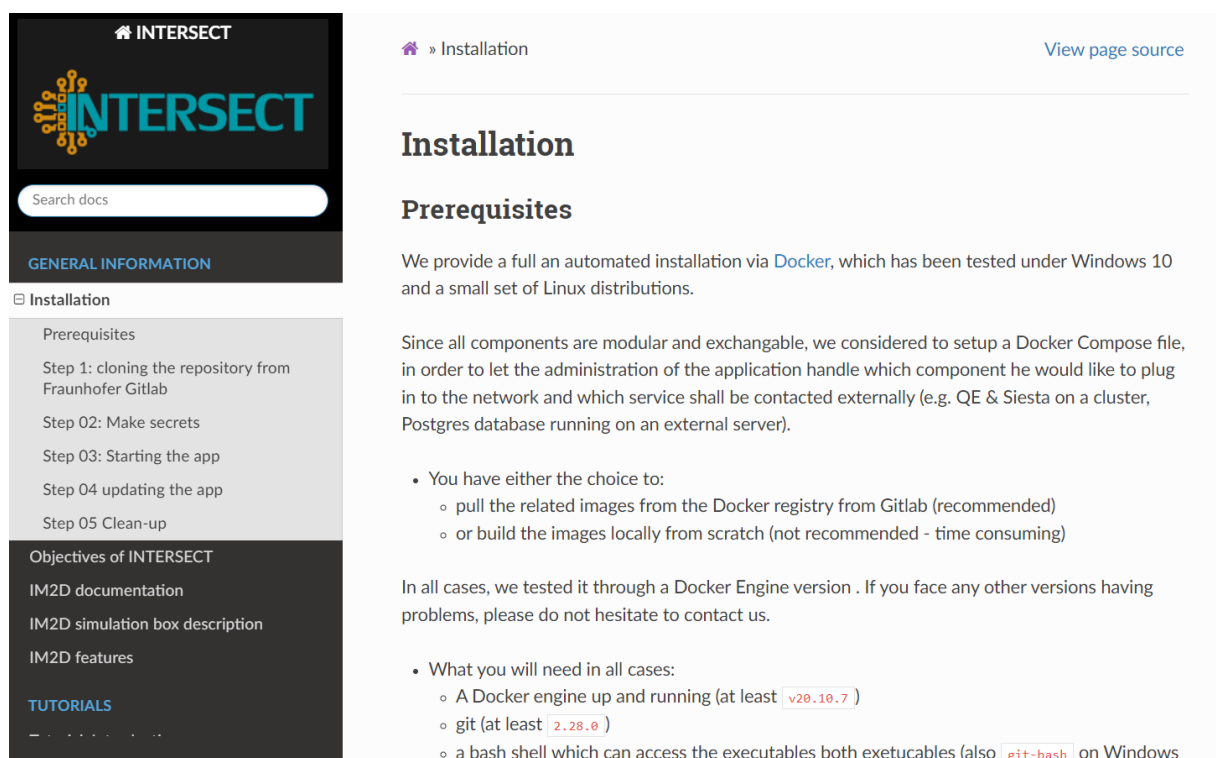


Figure 9: Fraunhofer Gitlab repository for managing the content of the documentation under <https://gitlab.cc-asp.fraunhofer.de/intersect/docs>.

7.2.1 Installation guide

One of the most important contents of the documentation is the installation guide for the backend services of IM2D, which are the main drivers of the simulations and workflows, as well as the source of data and knowledge for each workflow and its input/output data. As shown in Figure 10, the guide reports the software-requisites expected on the host system and the associated versions for which the app was tested.

Furthermore, each step for the cloning of the *aiida-wrapper* repository, configuration and execution of the Docker-containers is explained with code snippets for the terminal. Depending on the operating system, different notes and warnings for troubleshooting were added to the single steps as shown in Figure 11.



The screenshot shows the INTERSECT documentation website. On the left is a sidebar with a search bar and a navigation menu. The main content area is titled 'Installation' and includes a 'Prerequisites' section. The sidebar menu includes: INTERSECT, Search docs, GENERAL INFORMATION, Installation (selected), Prerequisites, Step 1: cloning the repository from Fraunhofer Gitlab, Step 02: Make secrets, Step 03: Starting the app, Step 04 updating the app, Step 05 Clean-up, Objectives of INTERSECT, IM2D documentation, IM2D simulation box description, IM2D features, TUTORIALS, and a list of links.

Installation

Prerequisites

We provide a full an automated installation via [Docker](#), which has been tested under Windows 10 and a small set of Linux distributions.

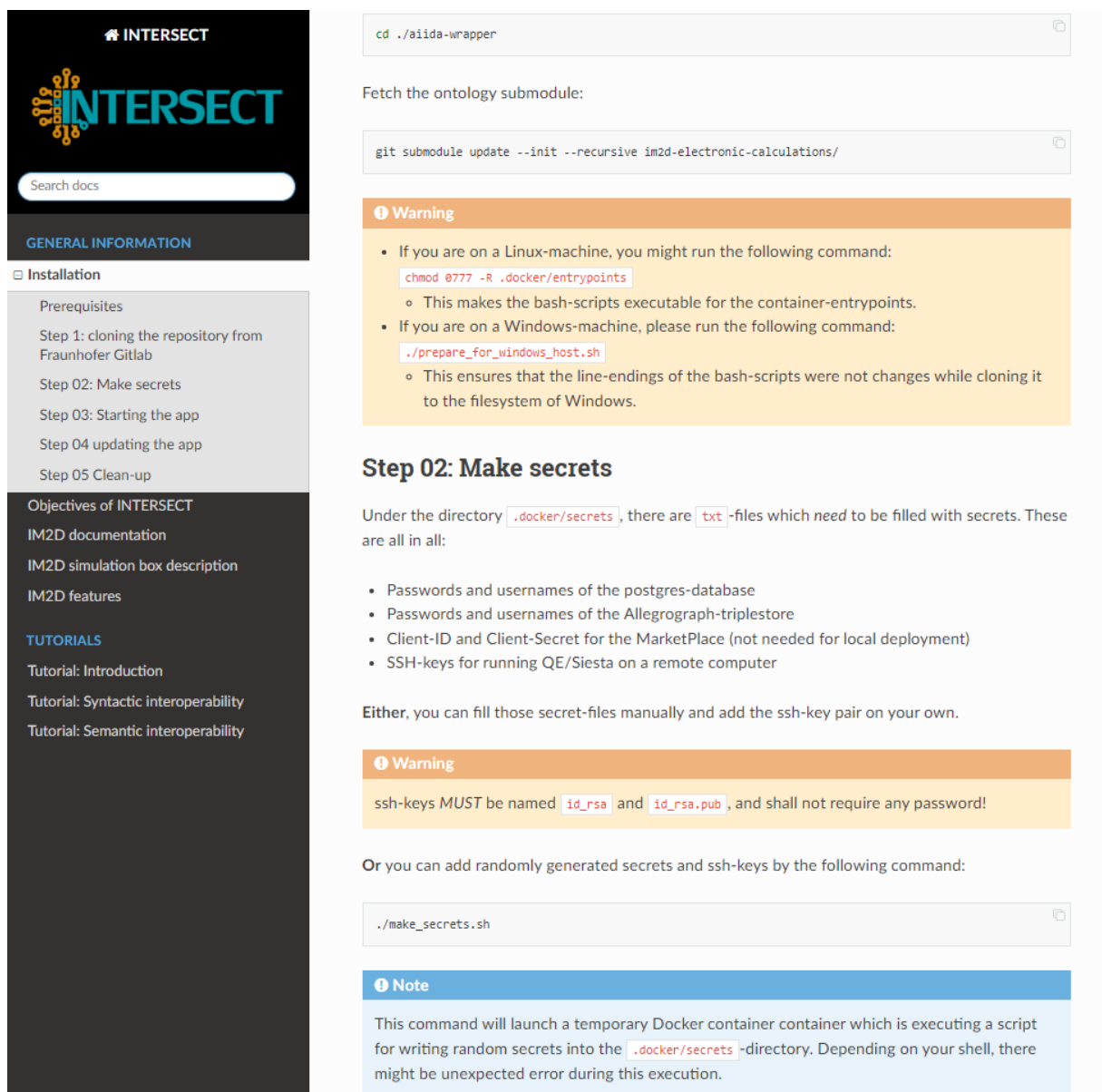
Since all components are modular and exchangeable, we considered to setup a Docker Compose file, in order to let the administration of the application handle which component he would like to plug in to the network and which service shall be contacted externally (e.g. QE & Siesta on a cluster, Postgres database running on an external server).

- You have either the choice to:
 - pull the related images from the Docker registry from Gitlab (recommended)
 - or build the images locally from scratch (not recommended - time consuming)

In all cases, we tested it through a Docker Engine version . If you face any other versions having problems, please do not hesitate to contact us.

- What you will need in all cases:
 - A Docker engine up and running (at least `v20.10.7`)
 - git (at least `2.28.0`)
 - a bash shell which can access the executables both exetucables (also `git-bash` on Windows

Figure 10: Explanation of prerequisites for IM2D installation.



INTERSECT

Search docs

GENERAL INFORMATION

Installation

Prerequisites

Step 1: cloning the repository from Fraunhofer Gitlab

Step 02: Make secrets

Step 03: Starting the app

Step 04 updating the app

Step 05 Clean-up

Objectives of INTERSECT

IM2D documentation

IM2D simulation box description

IM2D features

TUTORIALS

Tutorial: Introduction

Tutorial: Syntactic interoperability

Tutorial: Semantic interoperability

```
cd ./aiida-wrapper
```

Fetch the ontology submodule:

```
git submodule update --init --recursive im2d-electronic-calculations/
```

Warning

- If you are on a Linux-machine, you might run the following command:


```
chmod 0777 -R .docker/entrypoints
```

 - This makes the bash-scripts executable for the container-entrypoints.
- If you are on a Windows-machine, please run the following command:


```
./prepare_for_windows_host.sh
```

 - This ensures that the line-endings of the bash-scripts were not changes while cloning it to the filesystem of Windows.

Step 02: Make secrets

Under the directory `.docker/secrets`, there are `txt`-files which *need* to be filled with secrets. These are all in all:

- Passwords and usernames of the postgres-database
- Passwords and usernames of the Allegrograph-triplestore
- Client-ID and Client-Secret for the MarketPlace (not needed for local deployment)
- SSH-keys for running QE/Siesta on a remote computer

Either, you can fill those secret-files manually and add the ssh-key pair on your own.

Warning

ssh-keys *MUST* be named `id_rsa` and `id_rsa.pub`, and shall not require any password!

Or you can add randomly generated secrets and ssh-keys by the following command:

```
./make_secrets.sh
```

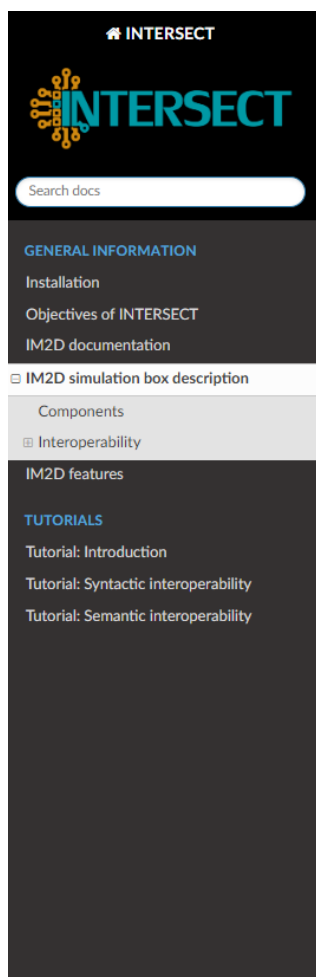
Note

This command will launch a temporary Docker container which is executing a script for writing random secrets into the `.docker/secrets`-directory. Depending on your shell, there might be unexpected error during this execution.

Figure 11: Screenshot of the installation steps with suggestions for troubleshooting.

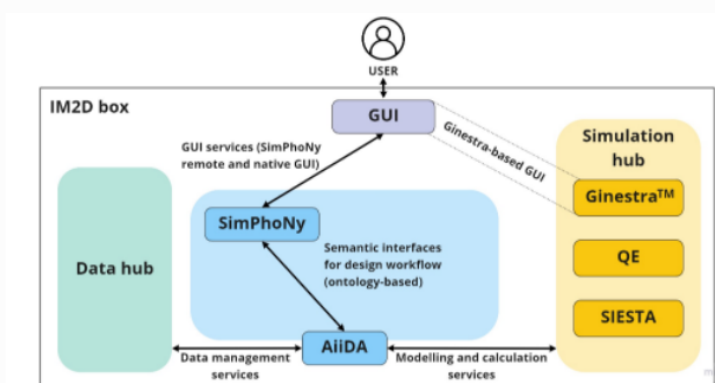
7.2.2 Description of properties, functionalities and ontologies

Technical description of IM2D architecture, GUI visualisation tool, and information about the project background (such as the participating consortium partners and the project objectives) were included in separate sections (Figure 12).



Components

The figure below shows a schematic overview of the IM2D components and their contribution to the IM2D simulation box: The graphical user interface (GUI), the semantic and syntactic workflow managers provided by SimPhoNy and AiiDA, respectively, the data hub, as well as the simulation hub consisting of the modelling and calculation services Quantum Espresso (QE) and Siesta.



Components of the IM2D simulation box.

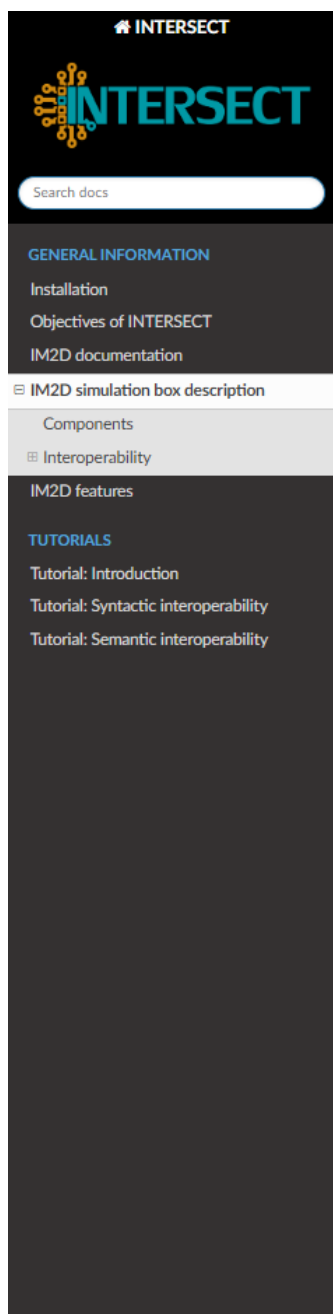
Graphical user interface:

- The Ginestra® GUI is the front-end of the IM2D platform, through which the software end user interacts with the resources and workflows provided by the REST APIs of SimPhoNy and AiiDA. The interface has an intuitive and user-friendly structure based on an operation menu, buttons for the connection with the other parts of the infrastructure (e.g., AiiDA), and windows for the visualization of 3D structures and material properties/parameters. It represents the Ginestra®-AiiDA plugin front-end and allows the semantic definition of different user profiles (explained in Chapter 3.2).
- Via pipelines the GUI provides a seamless integration of AiiDA, Ginestra®, and SimPhoNy, as well as the access to the data hub through the OPTIMADE API, as shown below.
- For more information, have a look at the INTERSECT deliverables D1.5 - GUI design and setup and D1.6 - GUI deployment.

Figure 12: Overview of the software architecture of IM2D and the role of the graphical user interface.

The subsequent paragraphs of the guide, describes the semantic interoperability architecture of IM2D by referencing the corresponding HTTP-queries and triggering the dedicated SPARQL-queries for the ontology through SimPhoNy. Additionally, interoperability for the submission of jobs directly through syntactic data (in JSON-serialisation) or semantic data (JSON-LD-serialisation of CUDS) is discussed by the model of the interoperability layer (Figure 13).

Furthermore, the distinction between the generic domain-ontology concepts of EMMO-crystallography and the mapping to the application-related concepts of IM2D, internally used by the *aiida-wrapper*, is covered in the text (Figure 14). Visualised with screenshots of Protégé, this fosters the understanding of (i) how the interface of SimPhoNy and AiiDA facilitates the knowledge graph in order to semantically describe the properties that are computed through the available workflows, (ii) how the inputs are filtered by SimPhoNy for different persona, and (iii) how the mapping to the data resources in the AiiDA-backend is achieved.

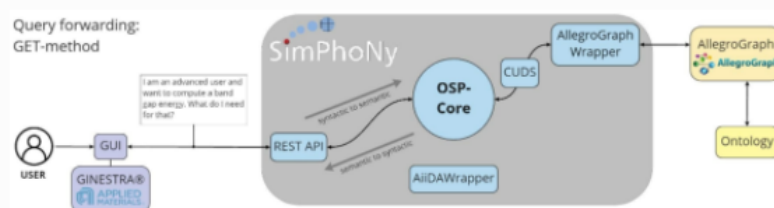


Example: GET-method

A nice example to demonstrate IM2D interoperability is the **GET-method**. The GET-method includes exchange of semantic and syntactic data between the GUI, SimPhoNy, AllegroGraph (a triple store), the ontology. In general, the first task that SimPhoNy has to handle when a user connects to the GUI is to supply selectable **user profiles**: basic, intermediate, and advanced. The individual requirements for each of the three user types differ depending on each expertise level ("Knowledge"), which is realized through the control of technical input parameters that affect the accuracy of the results (e.g. energy cutoffs, number of K-points, convergence thresholds, etc.). So, in the GUI, the user selects his/her user profile and enters the name of a material property of interest. Both conditions are embedded into an http-route in the following format:

- localhost:8000/api/v4/intersect/properties/my-property-of-interest/inputs/my-user-level

When this route is taken via GET-method in the SimPhoNy-REST service, the OSP-core sends a **SPARQL-query** to the **AllegroGraph** triple store, where ontology-based concepts are stored as triples.

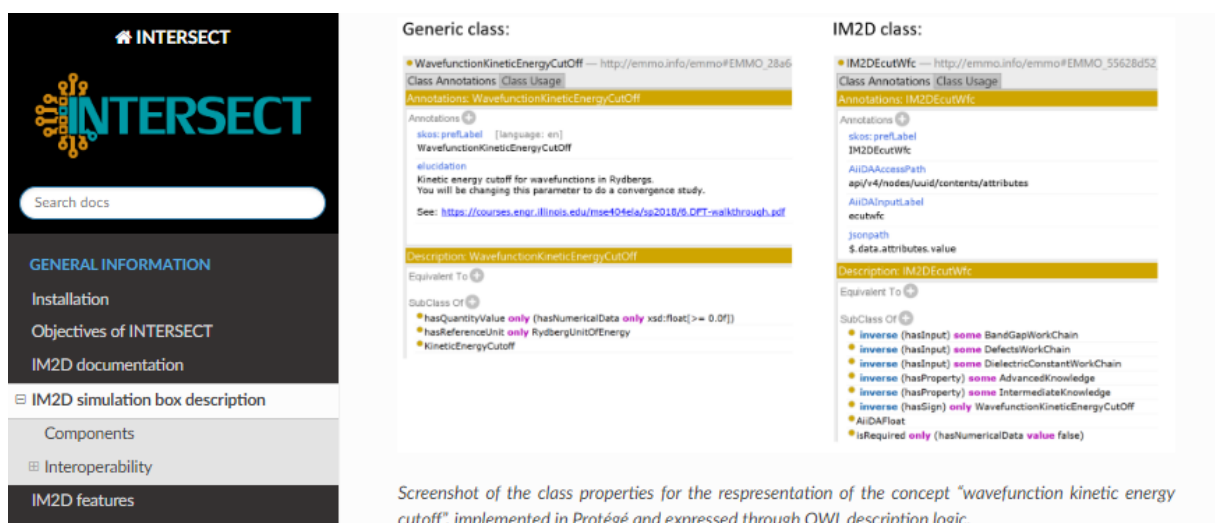


Placeholder: Query forwarding via GET-method: In the GUI, the user selects a user level and the material property of interest, e.g., the bandgap energy. The GUI sends a query to SimPhoNy to get a list of properties that are required as input parameters for the calculation of the demanded material property. The REST-API triggers the OSP-core to filter the required parameters from the ontology (saved in AllegroGraph as triples). From the ontology, CUDS objects are generated, carrying the semantic information of the query parameters. On the way back to the GUI, the semantic data is converted to syntactic data and the user receives a profile-dependent set of parameters required for the calculation of the bandgap energy.

The SPARQL-query is dynamically adjusted with respect to the specified user profile and AiiDA-workflow and gets a set of required parameters as a response, along with recommendation values and information about the data range-restrictions, data type-restrictions, unit-descriptions, unit-expressions and additional user advices such as further reading references. These are then transferred back as **http-response** from the SimPhony-REST API to the GUI. Some of the strings in this response, e.g. units and physical dimensions, are implemented in latex-syntax so to be displayed in the GUI. At this point, it is suitable to mention that the **ontology** carries all information necessary for the realisation of everything described above. The figure below provides an example for the ontological representation of material parameters in Protégé. On the one hand, it shows the distinction between the **generic** and **application-related IM2D** class and, on the other hand, how the corresponding IM2D class is restricted to certain workchains, user expertise ("Knowledge"), and AiiDA-related inputs.

Figure 13: Explanation of the semantic interoperability for getting input information for a certain workflow and user-level. Figure originating from D2.5 (Semantic interoperability of the automated workflows through SimPhoNy).

In the near future, this section will be further extended in order to deliver the knowledge on how to expand the collection of properties in IM2D to new codes and models. In particular, this should enable future IM2D developers to include new AiiDA-workflows via the IM2D-ontology, add new input parameters or even add new persona levels.




The screenshot displays the INTERSECT documentation website on the left and two Protégé class property windows on the right. The website features a search bar and a sidebar with navigation links: GENERAL INFORMATION, Installation, Objectives of INTERSECT, IM2D documentation, IM2D simulation box description, Components, Interoperability, and IM2D features. The Protégé windows show the class properties for 'WavefunctionKineticEnergyCutOff' (Generic class) and 'IM2DEcutWfc' (IM2D class). The Generic class window includes annotations for 'skos:prefLabel' and 'rdfs:label', a description of the kinetic energy cutoff, and a list of sub-classes. The IM2D class window includes annotations for 'skos:prefLabel' and 'rdfs:label', a description of the IM2D class, and a list of sub-classes.

Figure 14: Introduction to the ontologisation of the concepts and workflows facilitated in IM2D by coding examples in Protégé.

7.2.3 User tutorials

In order to provide code examples for a deeper understanding of the interoperability managed via the REST-API, a small set of JupyterNotebooks was included in the documentation (Figure 15). These show Python-code examples that use the **OSP**-core package, the **EMMO**, and the **DCAT**-, and IM2D-ontologies to interact with the available workflows, both syntactically and semantically. The corresponding REST-server required for IM2D can therefore be accessed either on the localhost (when IM2D is launched on a local machine) or on a remote server (such as the MarketPlace).



GENERAL INFORMATION

Installation

Objectives of INTERSECT

IM2D documentation

IM2D simulation box description

IM2D features

TUTORIALS

Tutorial: Introduction

Tutorial: Syntactic interoperability

Tutorial: Semantic interoperability

Now, create some CUDS-object and do the mapping with dcat and emmo!

```
[6]: from osp.core.namespaces import emmo, dcat2, dcterms
      from uuid import UUID

      med = dcterms.MediaType(iri="https://www.iana.org/assignments/media-types/application/json")

      unitcell = emmo.UnitCell(uuid=UUID(uuid))
      rec = dcat2.Resource(iri=f"{url}?jsonpath=${$.data.attributes.struc}")
      unitcell.add(med, rel=dcat2.mediaType)
      unitcell.add(rec, rel=dcat2.downloadURL)

      wfc = emmo.WavefunctionKineticEnergyCutOff()
      rec = dcat2.Resource(iri=f"{url}?jsonpath=${$.data.attributes.wfc}")
      wfc.add(med, rel=dcat2.mediaType)
      wfc.add(rec, rel=dcat2.downloadURL)

      rho = emmo.ChargeDensityKineticEnergyCutOff()
      rec = dcat2.Resource(iri=f"{url}?jsonpath=${$.data.attributes.rho}")
      rho.add(med, rel=dcat2.mediaType)
      rho.add(rec, rel=dcat2.downloadURL)

      emmo.OrdinaryGaussianSpreading()
```

[6]: <dcate.Resource: http://localhost:8000/api/v4/nodes/03a0b57b-6a8a-49ca-959c-c1182533f9f7/co

Now let us submit push the CUDS in order to calculate band gap:

```
[ ]: from osp.core.utils.general import (
      _serialize_session_json, _deserialize_cuds_object
      )

      url = "http://localhost:8000/api/v4/intersect/submit/band_gap.pw"

      serialized_session = _serialize_session_json(unitcell.session)
      headers = {"Content-type": "application/ld+json"}
      response = requests.post(url, headers=headers, data=serialized_session)
```

Receive the CUDS once the calculation is finished:

```
[ ]: uid = "03a0b57b-6a8a-49ca-959c-c1182533f9f7"

      url = f"http://localhost:8000/api/v4/intersect/cuds/{uid}"

      headers = {"accept": "application/ld+json"}

      response = requests.get(f"{url_cuds}/{status_uid}", headers=headers)
      cuds = _deserialize_cuds_object(response.json)
```

Figure 15: Code snippets for the computation of a band gap through QE, taken from the tutorials for teaching semantic interoperability on the IM2D documentation.

7.3 Swagger UI

The Swagger UI is a novel type of documentation of REST-APIs and was introduced by SmartBear Software Inc. By the simple provision of an openAPI-yaml-file (which had already been implemented for IM2D in D1.6 – GUI deployment), a compatible Python-package (like Flasgger) can interpret this markup-file and automatically generate a graphical interface. This interface exhibits all potential routes of a REST-service to be approached. Furthermore, it specifies which kind of data types are allowed as potential variables and inputs for certain parameters in the URL, keyword arguments in the HTTP-query, or serialisations in the HTTP-body. Additionally, the user is able to directly interact and test the endpoints with given

examples in any common web-browser without the need to enter a separate programming environment (see <https://app3.materials-data.space> as examples for IM2D hosted on the MarketPlace). Moreover, an equivalent cURL-command is generated, which can directly be used in any Linux-command line. The possible output formats are displayed with examples as well. Figure 16 shows a screenshot of some of the available HTTP-endpoints of the IM2D app visualised through the Swagger UI.

Overall, a Swagger UI offers the opportunity for any software developer to directly inspect the backend services of the IM2D-application and directly integrate its capabilities into their own application – regardless of the driving programming language.

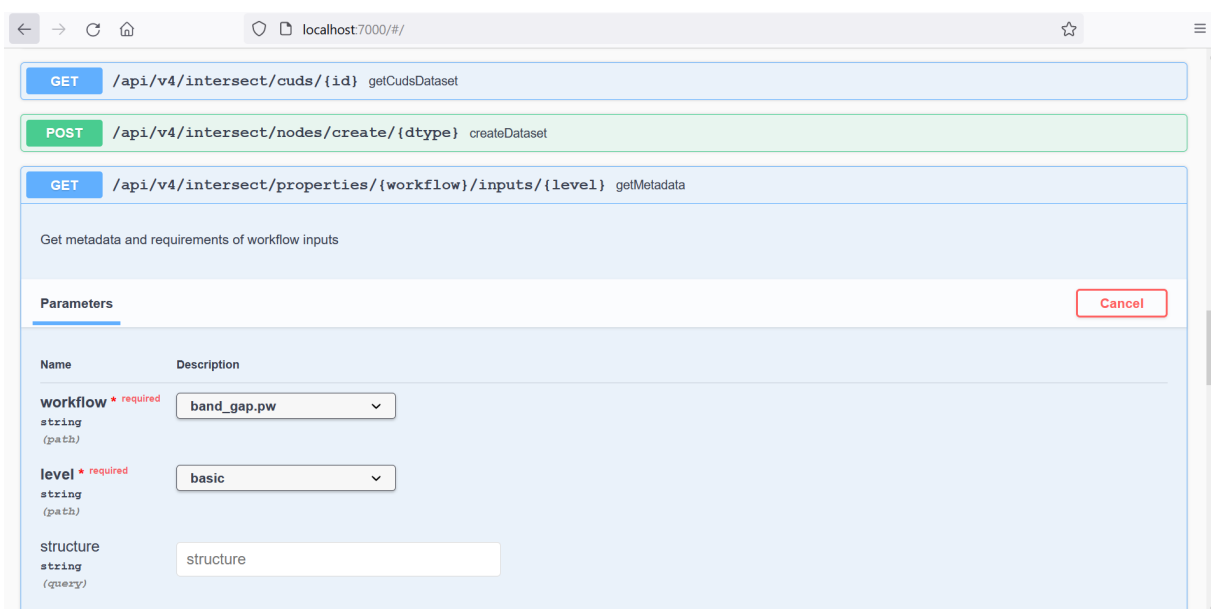


Figure 16: Screenshot of some of the available HTTP-endpoints of the IM2D app visualised through the Swagger-UI. The fields of "workflow" and "level" are variables in the URL which can be replaced with different options for a workflow to be run and with the user level to be chosen. The dropdown-menus exhibit all available options for those variables. The return of this HTTP-request is a JSON-file holding all potential input parameters of a certain workflow the user may adjust for a demanded user level. The field "structure" is a placeholder for a material-UUID for which the user wants to have a recommendation for default input values for different simulation accuracies. The URL from this REST-endpoint is then appended to the specified host the application is running on [10] and can be used by any third-party-tool such as the GUI.

For the case of semantic data, the programmer is able to download CUDS-objects as output of the simulation in any common serialisation format (TTL/OWL/JSON-LD) and to directly integrate it into any third-party tool by using ontologies (e.g., other SimPhoNy-wrappers, Protégé, WebVOWL, etc.).

Furthermore, SPARQL-queries can be inserted manually at a separate route in order to semantically explore the available data for certain materials and properties defined by the EMMO-ontologies (see Figure 17). In case of a non-SPARQL-user, the programmer can also search individuals of a specific property for the specific EMMO-UUIDs of an ontology-class (e.g., *EMMO_b2f5be57-53bb-4971-8365-681cc2024a47* for the band gap) or search by the associated label of that class (e.g., *BandGap*, *UnitCell*, etc.).

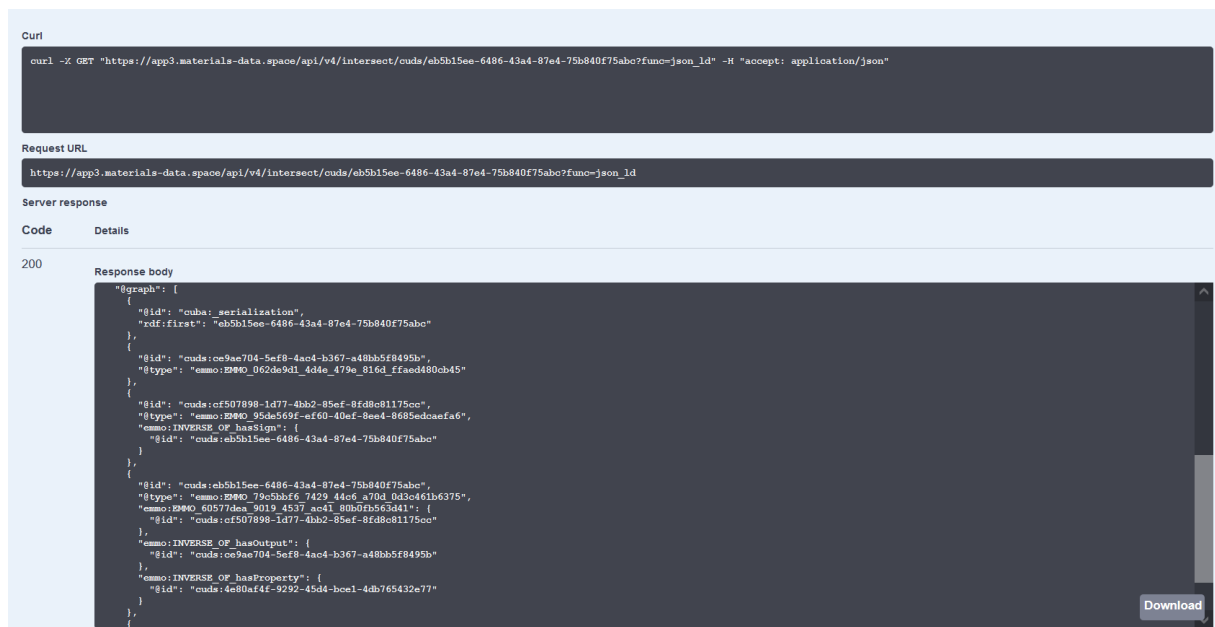


Figure 17: Examples for a response when the Swagger UI is used for querying for a specific CUDS (here: `eb5b15ee-6486-43a4-87e4-75b840f75abc` for the unit cell of silicon). The upper window shows the corresponding cURL-command and the lower window the TTL-file serialisation which can be downloaded directly and used by any other third-party tool.

Conclusion and outlook

Considering the development activities for the source code of the IM2D toolbox, we put in place a set of modern technologies, such as git and its supporting tools like CI/CD-pipelines, for testing the code stability and ReadTheDocs documentation.

The repositories involved in the IM2D simulation box are hosted on the private Fraunhofer Gitlab and public Github. Since the versioning is related to the docker registry, the reproducibility of previous code variations is guaranteed. The pipelines are supported by different stages of unit tests, which check the range from basic functionalities of the REST API up to the performance of a whole workflow simulating material properties like the band gap. The documentation for the backend services of AiIDA and SimPhoNy comprises a broad range of contents, covering the installation guide, the description of the app architecture and the user tutorials for the REST API. The REST API itself is documented through the openAPI-specification that produces an interactive Swagger UI that can be used to explore all possible entry points and routes of the HTTP server.

The documentation for the corresponding GUI is available separately from the ReadTheDocs-pages in the form of a PDF and can be obtained by contacting the AMAT customer service or the INTERSECT coordination directly.



Potential future development steps may include further descriptions for the testing and/or the implementation of advanced configurations in the official documentation. This may, e.g., incorporate a possible scaling of the app components to other databases and triplestores, or connecting the workflow management of AiiDA to high-performance clusters or other associated facilitations. Regarding the documentation, the description of the application- and domain-ontologies need to be expanded so that further properties and workflows can be added to the semantically interpreted workflows of SimPhoNy and AiiDA. This would also enable the introduction of new input parameters, properties, workflows, and even new persona profiles. On the other hand, also the docker-compose setup will be documented in more detail so that an app-administrator is able to exchange single app components such as the QE- and SIESTA-version or the triplestores and databases without further effort.

References and links

- [1] <https://gitlab.cc-asp.fraunhofer.de/simphony/wrappers/aiida-wrapper>
- [2] <https://github.com/aiidateam/aiida-quantumespresso>
- [3] https://github.com/siesta-project/aiida_siesta_plugin
- [4] <https://github.com/epfl-theos/aiida-defects>
- [5] <https://github.com/aiidateam/aiida-pseudo>
- [6] https://gitlab.cc-asp.fraunhofer.de/intersect/ext_to_aiida
- [7] <https://gitlab.cc-asp.fraunhofer.de/simphony/wrappers/allegrograph-wrapper>
- [8] <https://gitlab.cc-asp.fraunhofer.de/ontology/applications/intersect/im2d-electronic-calculations>
- [9] <https://gitlab.cc-asp.fraunhofer.de/intersect/docs>
- [10] `localhost:7000/api/v4/intersect/properties/band_gap.pw/inputs/basic` in case of a local deployment or https://app3.materials-data.space/api/v4/intersect/properties/band_gap.pw/inputs/basic in case of the MarketPlace.

Acronyms

- CI – Continuous Integration
- CD – Continuous Deployment
- CUDS - Common Universal Data Structure
- EMMO – European Material Modelling Ontology
- GUI – Graphical User Interface
- IM2D – Interoperable Materials-to-Device
- REST API – Representational State Transfer Application Programming Interface
- OSP – Open Simulation Platform
- UI – User Interface