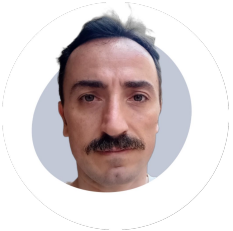
ELISA
Enabling Linux in Safety Applications

WORKSHOP

ELISA Workshop
Munich, Germany

November 18-20, 2025
Co-hosted with Red Hat

Who I am



Luigi Pellecchia

Principal Software Quality Engineer

In-vehicle OS
Red Hat

Member of Technical Steering Committee

ELISA
(Linux Foundation)

Agenda

- BASIL Overview
- What's new

ELISA Enabling Linux in Safety Applications WORKSHOP

BASIL The FuSa Spice

Tool developed to manage software related work items, design their traceability towards specifications and ensure completeness of analysis

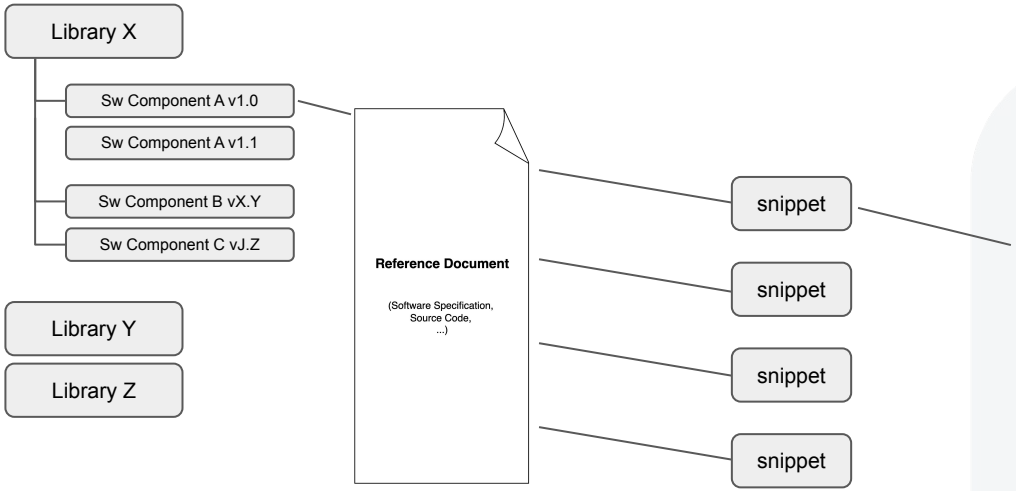
- Born at Red Hat to support RHIVOS Functional Safety ISO 26262 Compliance Certification
- BASIL name comes from ASIL B
- Presented to ELISA Project on June 2023 during the Berlin Workshop
- Open Sourced and hosted at ELISA github

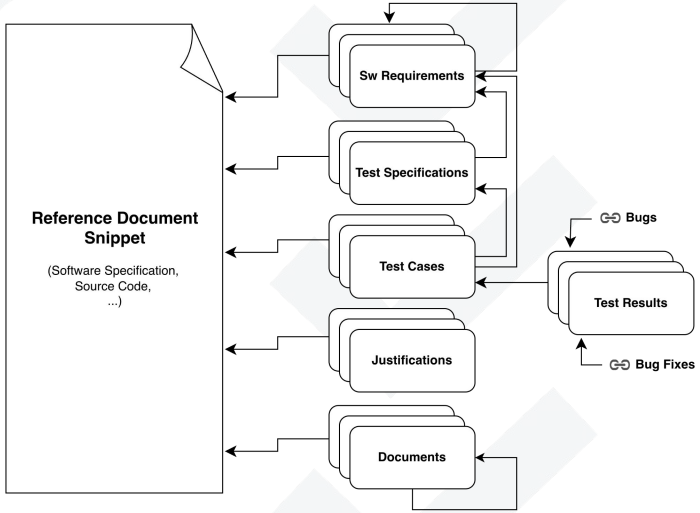
ELISA Enabling Linux in Safety Applications

WORKSHOP

BASIL The FuSa Spice

Define the traceability matrix by creating the work items





ELISA Enabling Linux in Safety Applications

WORKSHOP

BASIL - Strengths

- Open Source
- Web user interface and HTTP REST API
- Extended traceability and SPDX SBOM generation
- Keeps history of all changes
- Granular user permissions management
- Clarifies gaps and promotes collaboration
- In-app and email notifications
- Embedded test infrastructure and support for external test infrastructures
- Import Software Requirements (SPDX Model 3 json, yaml, json, csv, xlsx)
- Import Test Cases from remote test repositories (via tmt)

BASIL supported test Infrastructures

Test Infrastructure	Trigger and Trace	Trace pre existing runs	Type	Available starting from version
tmt	✅	❌	Embedded	>= 1.4
Gitlab CI	✅	✅	External	>= 1.5
GitHub Actions	✅	✅	External	>= 1.5
KernelCI	❌	✅	External	>= 1.5
Testing Farm	✅	❌	External	>= 1.5
LAVA (Linaro)	✅	✅	External	>= 1.7

What’s new?

Apache deployment

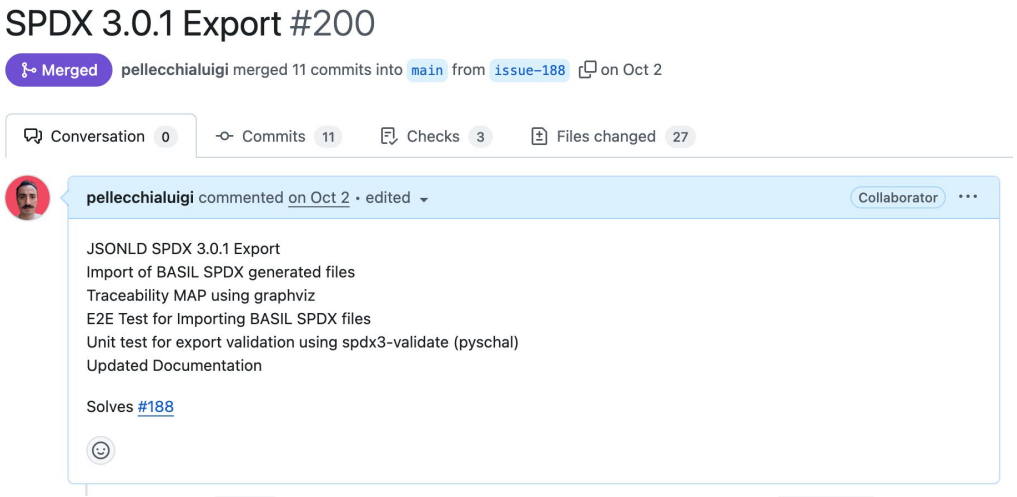
BASIL / deploy / **apache** /

Name
..
.env
README.md
build_basil_api.sh
build_basil_frontend.sh
common.sh
init_postgresql.sh
run.sh

```yaml
apache_deployment:
  runs-on: ubuntu-latest
  steps:

    - name: Checkout
      uses: actions/checkout@v4

    - name: Define global variables
      id: global_variables
      run: |
        echo "BRANCH_NAME=${{ github.head_ref || github.ref_name }}" >> "$GITHUB_OUTPUT"
        sed -i 's/^BASIL_TESTING=.*/BASIL_TESTING=1/' deploy/apache/.env
        sed -i 's/^BASIL_API_PORT=.*/BASIL_API_PORT=5005/' deploy/apache/.env
        sed -i 's/^BASIL_APP_PORT=.*/BASIL_APP_PORT=9056/' deploy/apache/.env
        sed -i 's/^BASIL_ADMIN_PASSWORD=.*/BASIL_ADMIN_PASSWORD=dummy_password/' deploy/apache/.env
        sed -i 's/^BASIL_DB_PASSWORD=.*/BASIL_DB_PASSWORD=dbSecret123/' deploy/apache/.env
        TEMPDIR=$(mktemp -d)
        sed -i "s|^BASIL_DB_PASSWORD=.*|BASIL_DB_PASSWORD=${TEMPDIR}|" deploy/apache/.env

    - name: Make deploy/apache/run.sh executable
      run: chmod +x deploy/apache/run.sh

    - name: Run the deployment
      run: |
        cd deploy/apache && sudo ./run.sh
```

SPDX Export Refactoring

- SPDX export implemented at version 1.6.x was generating files that are not complaints with the spdx 3.0.1 schema
- The first approach was based on spdx3 python module
- A complete refactor of the feature was needed to complies with the spdx 3.0.1 schema
- A CI test has been created to ensure BASIL exported files pass the schema validation



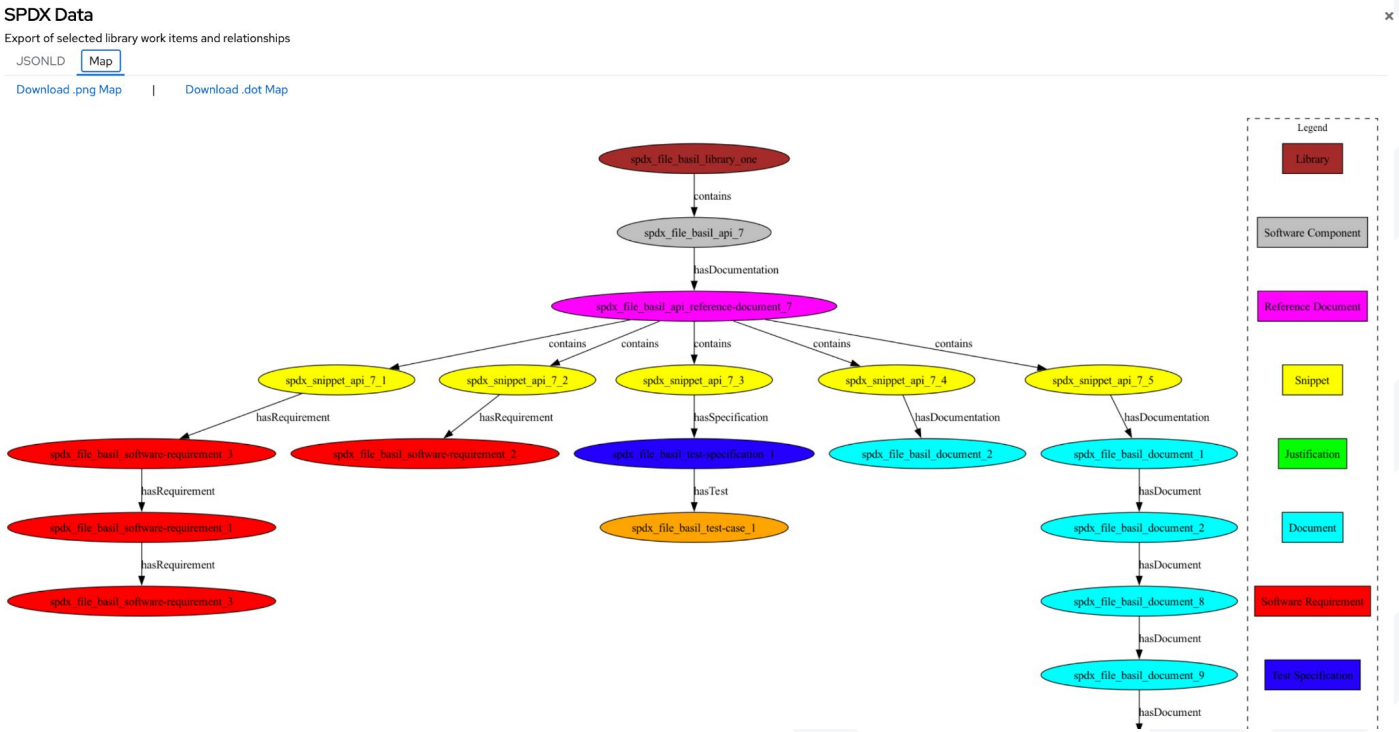
SPDX Export - CI Validation against spdx 3.0.1 schema

BASIL / api / test / test_spdx_api_validation.py

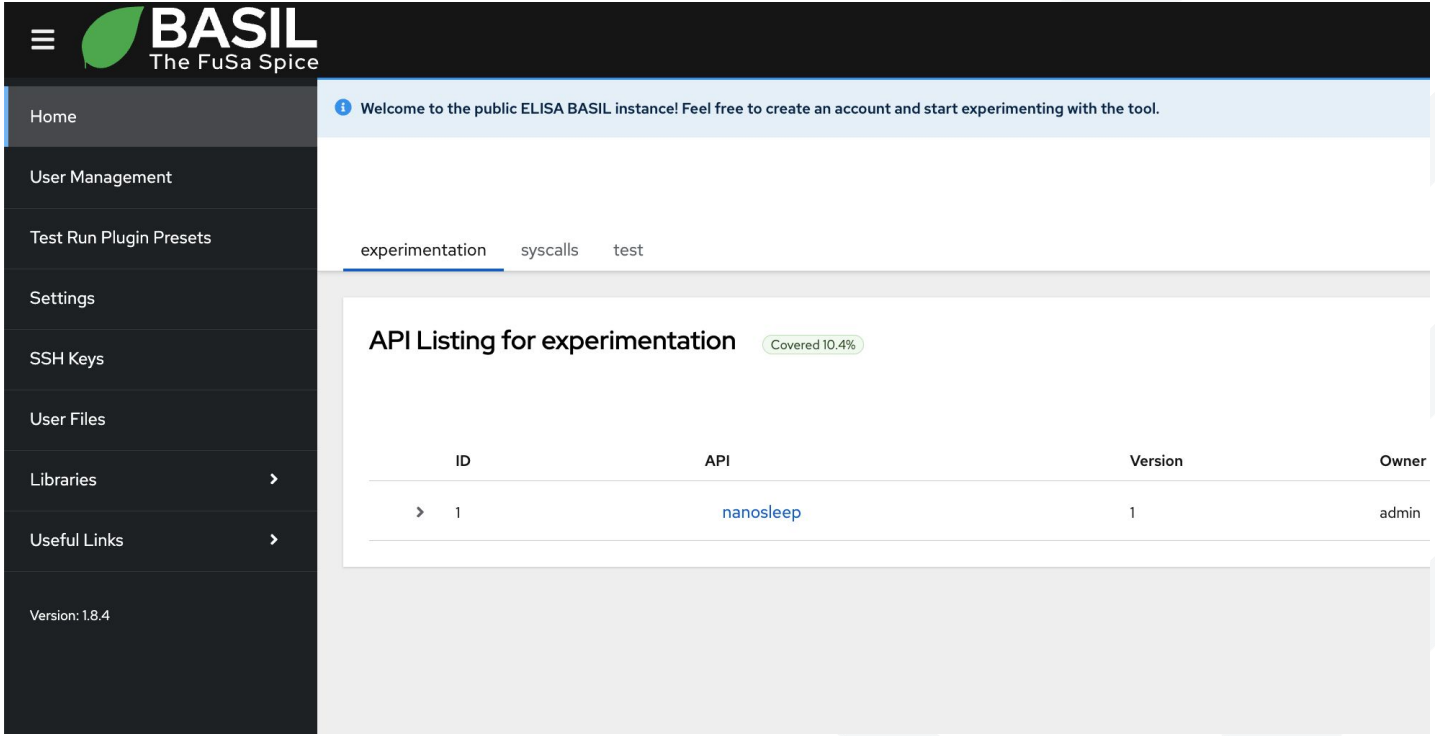
Code Blame 589 lines (497 loc) · 20.8 KB

```python
def test_spdx_api_export_and_validation(client, user_authentication, comprehensive_spdx_test_data):
    try:
        # Run spdx3-validate on the generated file
        print(f"Running spdx3-validate on {temp_file_path}")
        result = subprocess.run(
            ["spdx3-validate", "--json", temp_file_path, "--spdx-version", "auto"],
            capture_output=True,
            text=True,
            timeout=60,  # 60 second timeout
        )

        print(f"spdx3-validate exit code: {result.returncode}")
        if result.stdout:
            print(f"STDOUT: {result.stdout}")
        if result.stderr:
            print(f"STDERR: {result.stderr}")

        # Check validation results
        if result.returncode == 0:
            print("✓ SPDX validation passed successfully")
            assert True, "SPDX validation successful"
        else:
            print("✗ SPDX validation failed")
            # Don't fail the test immediately - let's analyze what went wrong
            validation_errors = result.stderr or result.stdout
            print(f"Validation errors: {validation_errors}")

            # You can add specific assertions here based on expected validation issues
            # For now, we'll record the failure but continue
            pytest.fail(f"SPDX validation failed with errors: {validation_errors}")
```

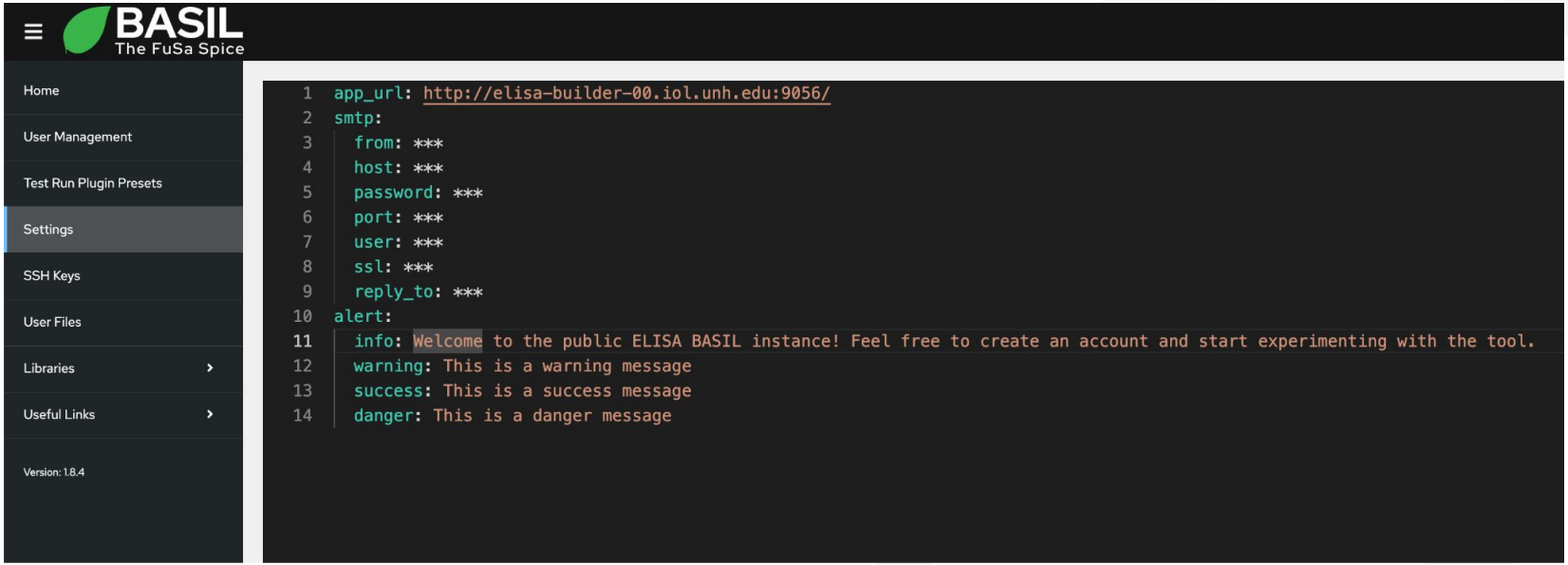
SPDX Export - Graph generation



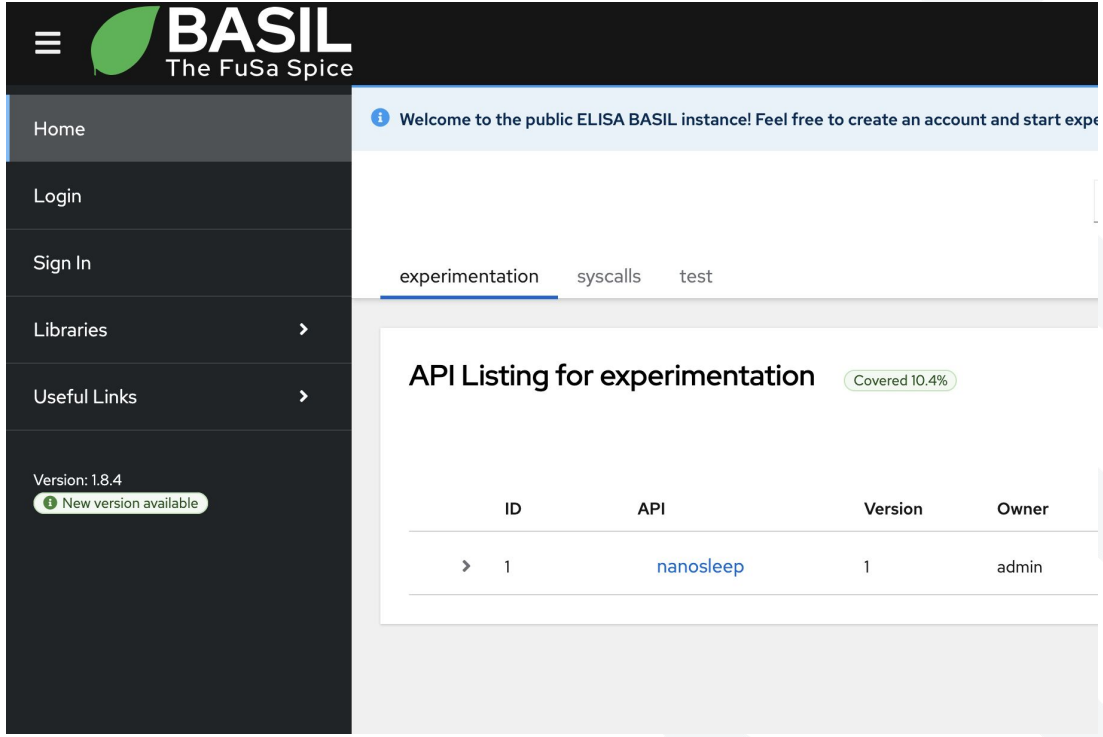
Configurable Alert Messages



Configurable Alert Messages



New Release available check



Demo on ELISA instance

Create an account and start experimenting with it

http://elisa-builder-00.iol.unh.edu:9056

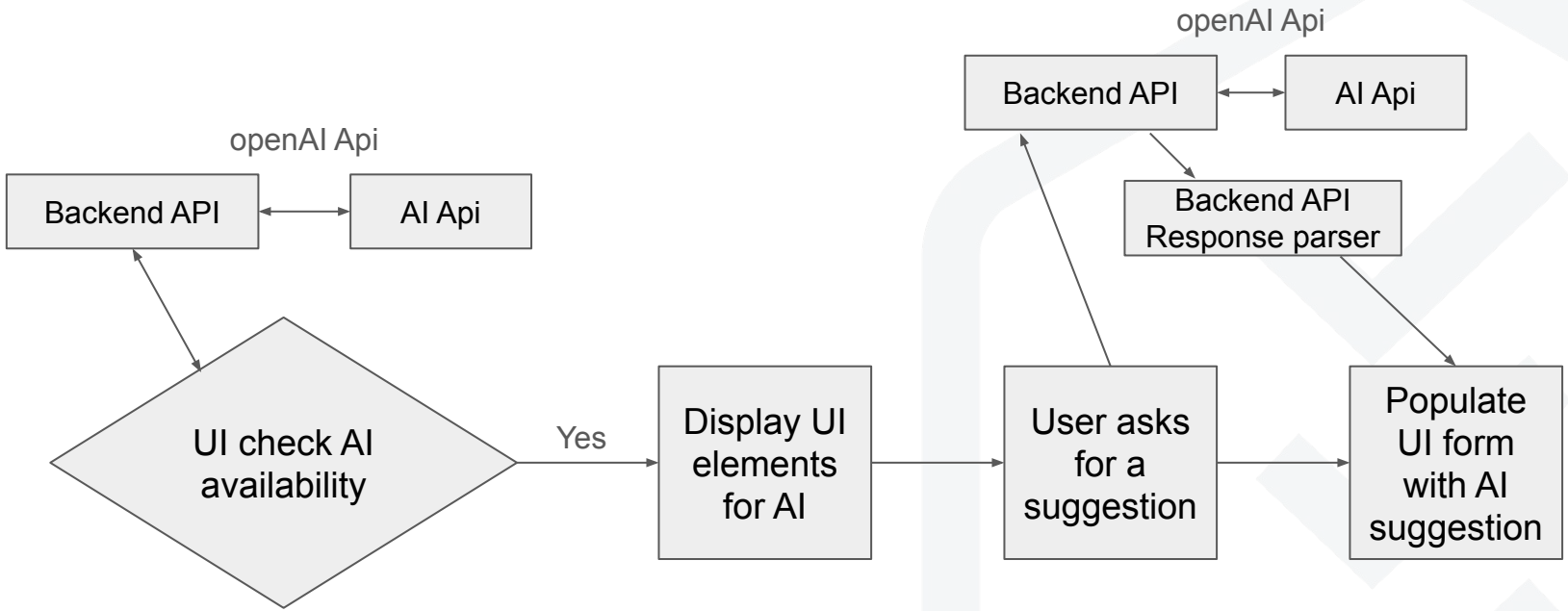
AI Support

- Admins can configure an external AI tool and communicate with it via openAI
- UI will enable components in case of successful connection with the AI endpoint

Can be used to:

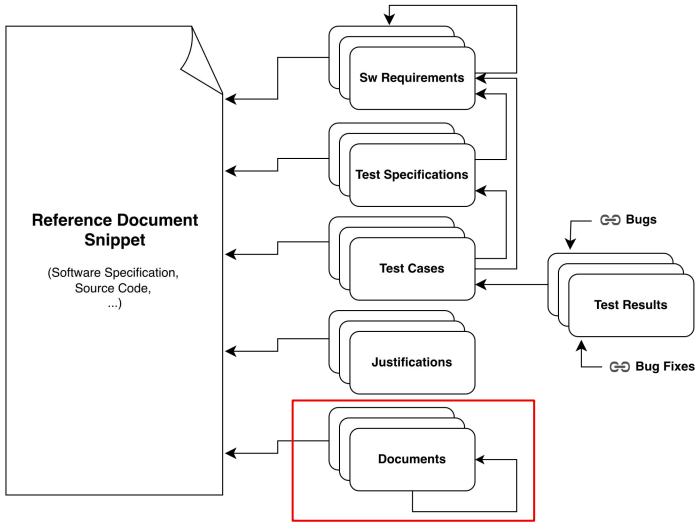
- Design work item from BASIL tool
  - Default prompts are asking to generate YAML like format that can be parsed and used to populate the UI in case of success
- Implement a Test Case based on Test Specification
  - Default prompts are asking to generate a file that will be store in the user files section

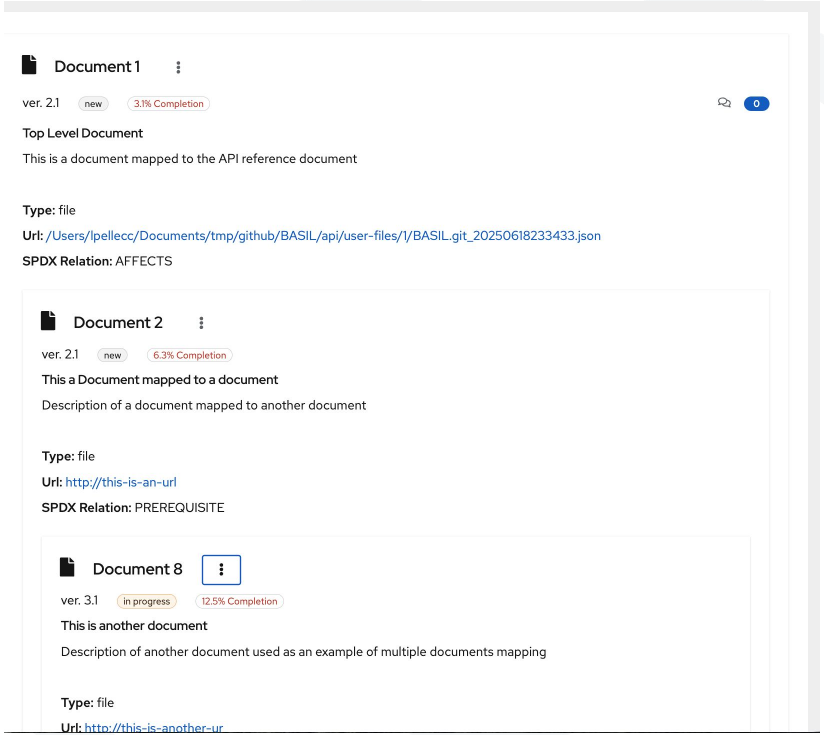
AI Support



Link to presentation at Open Source Summit Europe 2025
https://www.youtube.com/watch?v=17ftITXJjGI&t=2s&pp=ygUkYmFzaWwgd2hhdCdzIG9wZW4gc291cmNlIHN1bW1pdCAyMDI1

Hierarchical Documents Mapping





Traceability as Code

- Work items handled in git repositories
- Scan multiple external git repositories to extract data
- Shareable configuration file generates same results
- Automatically takes into account new work items
- Recreate the traceability of a target git commit
- Not tied to a single work item format
- Can be used in CI as the feature is provided by a command line tool
- Easy to extend with custom rules
- Can be used in CI automating the generation of an SBOM

Traceability as Code - Workflow

- Software components definition
- Repository configuration and candidate files extraction
- Reference document sections extraction
- Candidate work items data extraction per field
- Candidate work items data filtering
- Candidate work items data refactoring
- Traceability generation

Traceability as Code - Repository files filtering

Define rules to identify the files containing the work items data

- Is targeting a git commit/branch
- Reusable via YAML anchor
- Filter over
  - Files
  - Folders
  - File content

```yaml
repository: &repository_config
  url: "https://github.com/elisa-tech/BASIL.git"
  branch: "main"
  filename_pattern: "*.c"
  folder_pattern: "*examples*"
  hidden: False
  file_contains: ["read_mem"]
  file_not_contains: []
```

Traceability as Code - Work item field data extraction

Define how each field should be populated identifying **start** and **end** rules that will be applied over the candidate files.

The match condition can generate list of sections.

Supports relational search with **closest** match in a target **direction**

Allows **split** of a section based on a **delimiter** to generate multiple matches

```
description:
  start:
    line_contains: " read_mem("
  closest:
    line_contains: "Function's expectations:"
      direction: "up"
  end:
    line_contains: "* Context"
  split:
    by: "\n*\n"
```

Traceability as Code - Work items candidates extraction

Define rules to extract work items data from identified candidate files:

Rules for each work item fields

- Magic variables
- Relational search
- Support constant values
- List generation from each result

```
software_requirements:
  rules:
    - name: "sr1"
      repository: *repository_config
      skip_top_items: 1
      title:
        value: |
          "Function expectation __software_requirement_index__"
      description:
        start:
          line_contains: " read_mem("
        closest:
          line_contains: "Function's expectations:"
            direction: "up"
        end:
          line_contains: "* Context"
        split:
          by: "\n*\n"
```

Traceability as Code - Work items candidates filtering

Define rules to filter over the identified candidate work items:

- contains
- not contains
- regex

```
software_requirements:
  rules:
    - name: "sr1"
      repository: *repository_config
      skip_top_items: 1
      title:
        value: |
          "Function expectation __software_requirement_index__"
        filter:
          contains: ["keep", "1"]
          Case_sensitive: false
```

Traceability as Code - Work items candidates refactoring

Define rules to refactor and format work items data:

- replace
- regex substitution
- uppercase
- lowercase
- left trim characters
- right trim characters
- global trim
- prefix
- suffix

```
software_requirements:
  rules:
    - name: "sr1"
      repository: *repository_config
      skip_top_items: 1
      title:
        value: |
          "Function expectation __software_requirement_index__"
        rstrip: “!?)]}.,;:”
        transform:
          - how: replace
            what: what_to_find
            with: replace_with
          - how: prefix
            value: “Requirement: ”
```

Traceability as Code - Work items generation

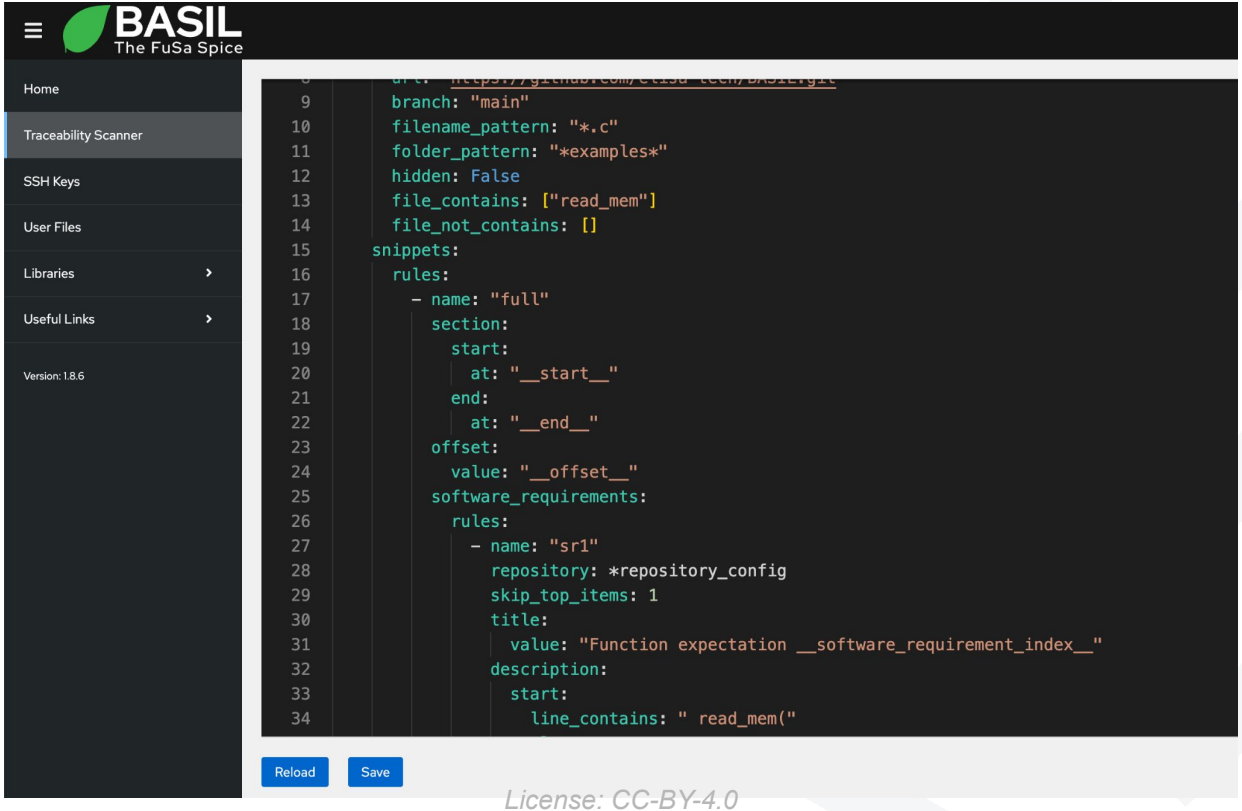
Check if the work items are already available in the BASIL database based on their content and create new ones if not.

Traceability as Code - Example

```
/**
* read_mem - read from physical memory (/dev/mem).
* @file: struct file associated with /dev/mem.
* @buf: user-space buffer to copy data to.
* @count: number of bytes to read.
* @ppos: pointer to the current file position, representing the physical
*        address to read from.
*
* This function checks if the requested physical memory range is valid
* and accessible by the user, then it copies data to the input
* user-space buffer up to the requested number of bytes.
*
* Function's expectations:
*
* 1. This function shall check if the value pointed by ppos exceeds the
*    maximum addressable physical address;
*
* 2. This function shall check if the physical address range to be read
*    is valid (i.e. it falls within a memory block and if it can be mapped
*    to the kernel address space);
*
.....
*/
 static ssize_t read_mem(struct file *file, char __user *buf,
                         size_t count, loff_t *ppos)
```

```yaml
software_requirements:

  rules:
  - name: "linux kernel requirements"
    repository: *linux_repo_config
     skip_top_items: 1
      title:
        value: "Function expectation
__software_requirement_index__"
      description:
        start:
          line_contains: " __api__("
          closest:
            line_contains: "Function's expectations:"
            direction: "up"
        end:
          line_contains: "* Context"
        split:
          by: "\n*\n"
        transform:
          - how: "regex_sub"
            what: " +"
            with: " "
          - how: "suffix"
            value: "."
        rstrip: ",.;:!? "
```

Traceability as Code - UI



Demo

Licensing of Workshop Results