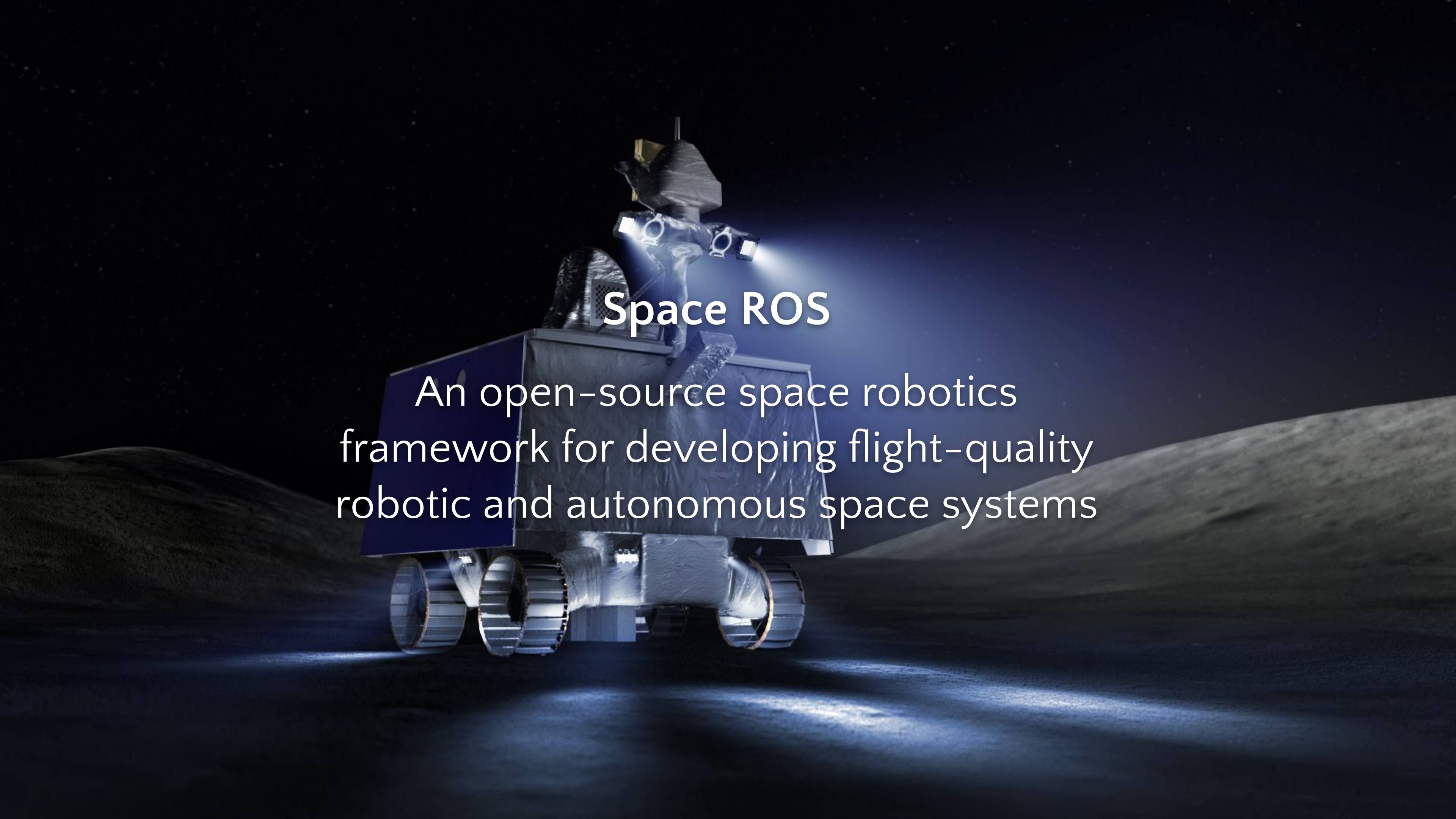
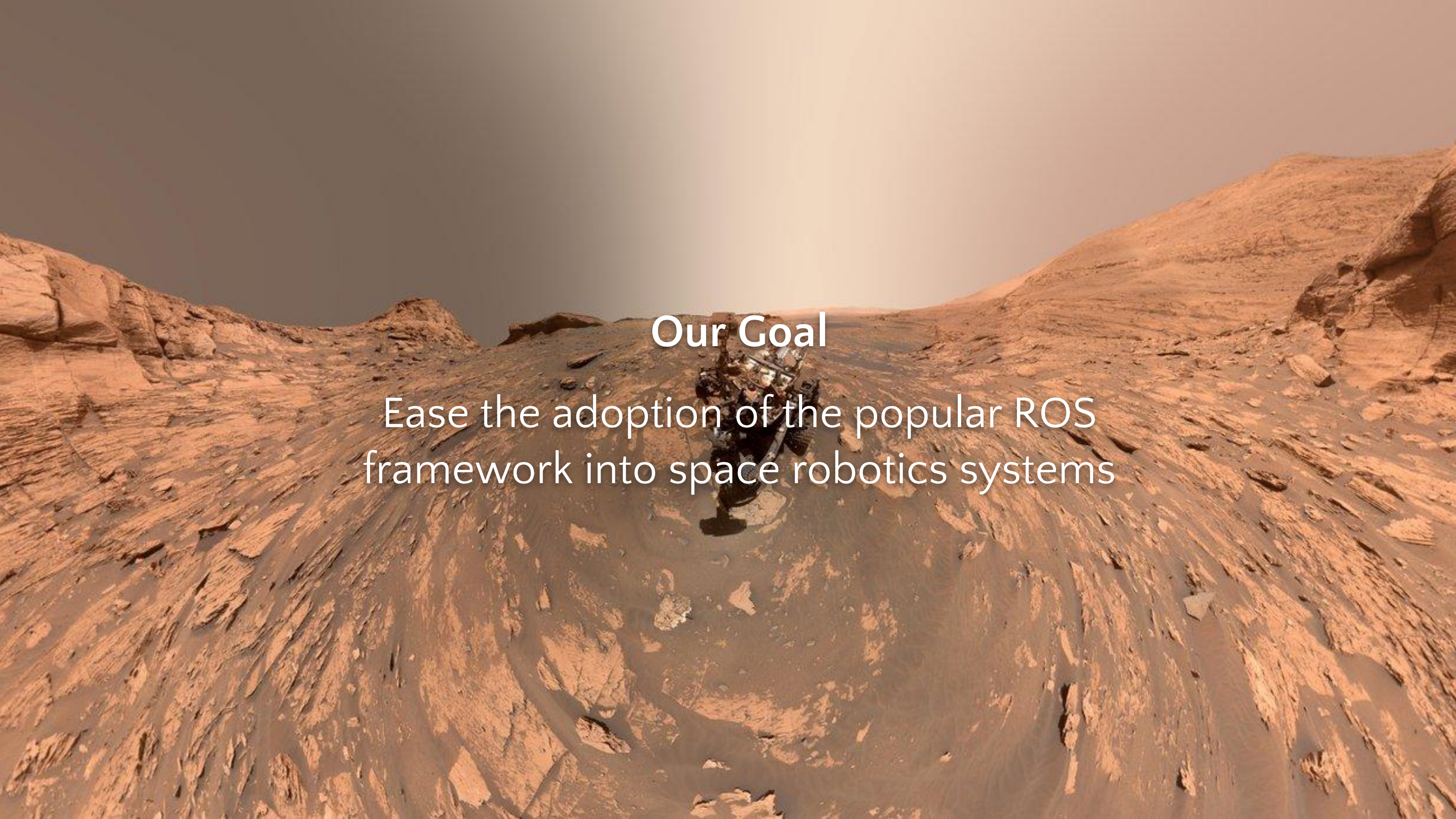


Space ROS

An open-source space robotics framework for developing flight-quality robotic and autonomous space systems



Our Goal

Ease the adoption of the popular ROS framework into space robotics systems

Certification-Ready

Provide software and artifacts that are aligned with aerospace standards

Open Source and Open Community

Bring the benefits of ROS to space robotics

spaceROS

A space-certifiable and reusable robotics framework

- Support certification to flight software standards, like DO-178C and NASA's NPR7150.2
- Provide artifacts to allow space flight projects to gain a head start on their certification efforts
- Aligned with NASA so that it can (eventually) be adopted for Class A missions
- Enable rapid development of new robotic capabilities
- Facilitate reuse across missions, reducing development effort and costs
- Based on open community, frameworks, and standards

What is Space ROS?

Space ROS 2022

Foundation

- Builds
- Releases
- Continuous Integration
- Maintenance
- Package subset
- Docker images

Tools and Processes

- Requirements tools and processes for traceability and analysis
- Code analysis tools with SARIF output
- Dashboard for issue navigation, visualization & dispositioning
- Development workflow
- Quality level(s)
- MC/DC testing

Space-Specific Functionality

- Eventing & Telemetry Subsystem
- C++ PMR allocator
- Sample applications
- Simulation assets
- Embedded target(s)

What is Space ROS?

Space ROS 2022

Foundation

- Builds
- Releases
- Continuous Integration
- Maintenance
- Package subset
- Docker images

Tools and Processes

- Requirements tools and processes for traceability and analysis
- Code analysis tools with SARIF output
- Dashboard for issue navigation, visualization & dispositioning
- Development workflow
- Quality level(s)
- MC/DC testing

Space-Specific Functionality

- Eventing & Telemetry Subsystem
- C++ PMR allocator
- Sample applications
- Simulation assets
- Embedded target(s)

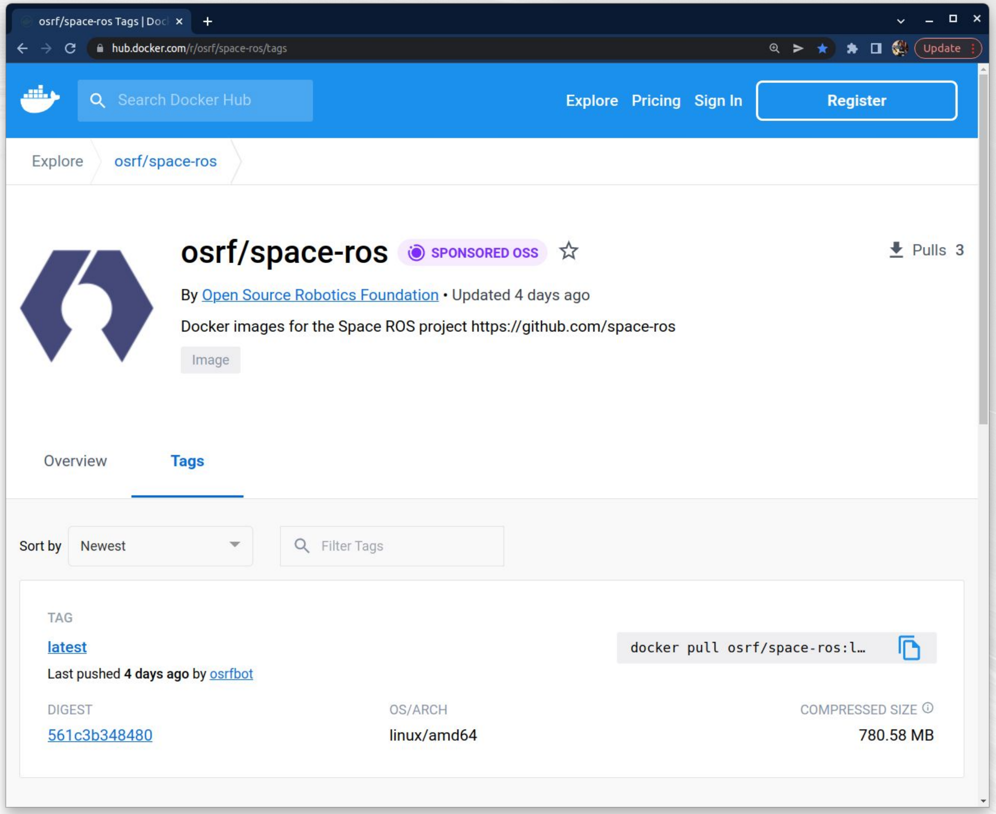
Foundation

Keeping things running smoothly

Item	Description
Space ROS GitHub Organization	The Space ROS source code is hosted in a dedicated GitHub organization, https://github.com/Space-ROS.
Automated Builds	The CI system builds the Space ROS source code at a specified frequency and run all unit tests.
Automated Build Output	Space ROS developers can use the Jenkins UI to view the output of each automated build, which includes any issues discovered by the code analysis tools.
Docker Images	Also, via github Actions, we are also building the Space ROS docker image nightly: https://github.com/space-ros/docker/actions. This helps us to catch any regressions in the build and helps us to keep the Docker image in good shape for prospective users.
Download of Space ROS Binaries	Space ROS developers can download Space ROS images that result from each (successful) automated build.

Foundation

Keeping things running smoothly



osrf/space-ros SPONSORED OSS

Pulls 3

By Open Source Robotics Foundation • Updated 4 days ago

Docker images for the Space ROS project https://github.com/space-ros

Image

TAG

latest

Last pushed **4 days ago** by osrfbot

```
docker pull osrf/space-ros:l…
```

DIGEST	OS/ARCH	COMPRESSED SIZE
561c3b348480	linux/amd64	780.58 MB

Foundation

Keeping things running smoothly

Item	Description
SARIF Viewing for all Code Analysis Tools	In addition to generating JUnit XML to integrate with Jenkins, the CI system generates SARIF output for each of the code analysis tools used for Space ROS. This can be either native output from the code analysis tools, or output that is then converted to the SARIF format.
Reporting of AUTOSAR C++ 14 Compliance Metrics	The CI System generates AUTOSAR C++ 14 compliance metrics for each Space ROS package.
Continuous Qualification System	Working towards an automated end-to-end Continuous Qualification System that collates all of the reports and artifacts.
Consistent Builds: CI and Local Environment	Using Earthly to provide a uniform environment between CI and local developer environments.

What is Space ROS?

Space ROS 2022

Foundation

- Builds
- Releases
- Continuous Integration
- Maintenance
- Package subset
- Docker images
- Embedded target(s)

Tools and Processes

- Requirements tools and processes for traceability and analysis
- Code analysis tools with SARIF output
- Dashboard for issue navigation, visualization & dispositioning
- Development workflow
- Quality level(s)
- MC/DC testing

Space-Specific Functionality

- Eventing & Telemetry Subsystem
- C++ PMR allocator
- Sample applications
- Simulation assets
- Embedded target(s)

Tools and Processes

TODO

Item	Description
Code Conformance Updates	Address issues identified by static analyzers (ongoing). Upstream changes.
Space ROS requirements process	Define a requirements process and associated tools for Space ROS.
Integrate Requirements Analysis Tool	Check requirements for consistency, conflicts, etc.
Integrate MC/DC Tool	Enable Modified Condition/Decision Coverage analysis. Required by DO-178C
Analyze ETS Requirements	Evaluate the requirements for the Eventing and Telemetry subsystem in order to prove out the methodology.
Requirements for an existing Space ROS package	Back-port requirements for a package, such as rcutils, and documentation on the process.
Space ROS Quality Levels	Define the Space ROS quality levels.

Requirements management in **aerospace**

More than checklists

- Typically managed using a strict process and proprietary tools
    - Process is often according to some accepted standard, e.g. DO-178C
- Requirements must be complete - no software without requirements - and highly detailed
- Multiple levels of requirements - from abstract needs to detailed behaviour timings
- Traceability is essential - source to requirement to implementation and verification, and back again
- Requirements ultimately are used to support a certification process

Requirements management in **open source software**

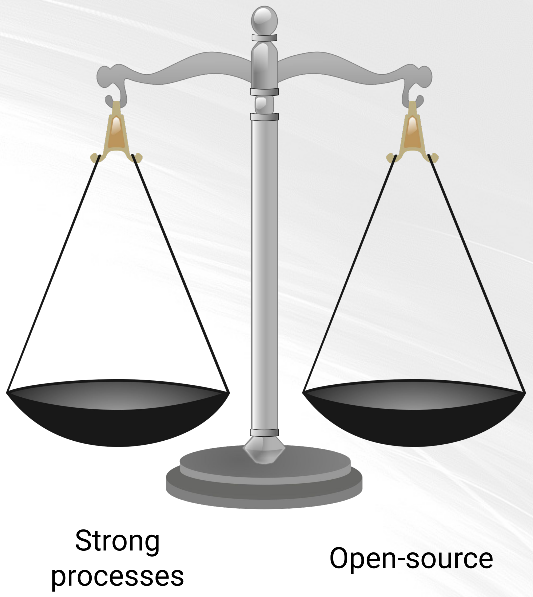
What requirements?

- Requirements are typically non-existent
- Any requirements that do exist are lightly managed (and easily get out-of-date)
- Heavy processes are shunned to avoid discouraging contributions

Open requirements for Space ROS

Balance competing forces

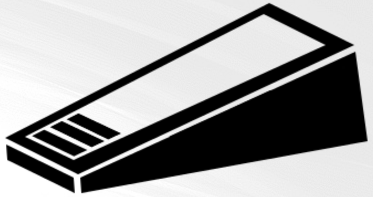
- Heavy-weight requirements process using expensive tools is inappropriate for an open-source project
- Need a process and tool(s) that won't discourage contributions
  - Contributors are unlikely to purchase expensive requirements management tools
  - Heavy-weight processes discourage drive-by contributors
- Must strike a balance between aerospace's need for strong processes and open-source's desire for ease-of-contributing



Tools for open requirements management

Doorstop

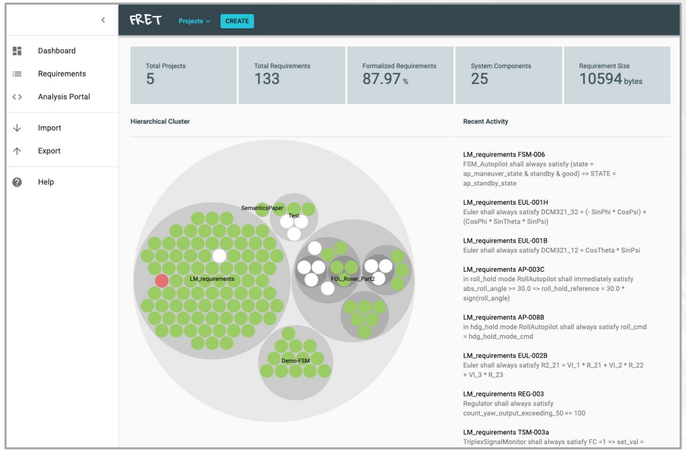
- Simple requirements management tool providing a command-line-and-text-editor based workflow
    - Add and edit requirements
    - Trace between requirements
    - Generate reports
- Based on YAML files stored in a versioned repository
    - Requirements are stored in a human-readable format
    - Easy to parse for additional automation tools
    - Requirements can be written in restricted natural language, e.g. EARS
- Open-source
    - Can be modified to meet our needs
    - Freely available to contributors
    - https://doorstop.readthedocs.io/en/latest/



Tools for open requirements management

FRET

- Graphical tool for creating and managing semi-formal and formal requirements
- Stores requirements in a database, with JSON import/export
- Requirements can be written in “FRETish”, which can contain linear temporal logic expressions
- Automatic model checking of requirements for consistency and conflicts
- Although freely available, the learning curve is steeper than Doorstop
- Automatic generation of safety monitor(s) from requirements expressed FRETish



Total Projects	Total Requirements	Formalized Requirements	System Components	Requirement Size
5	133	87.97 %	25	10594 bytes

Recent Activity

LM_requirements FSM-006
FSM_Autopilot shall always satisfy (state = ap_maneuver_state & standby & good) => STATE = ap_standby_state

LM_requirements EUL-001H
Euler shall always satisfy DCM321_32 = (- SinPhi * CosPsi) + (CosPhi * SinTheta * SinPsi)

LM_requirements EUL-001B
Euler shall always satisfy DCM321_12 = CosTheta * SinPsi

LM_requirements AP-003C
in roll_hold mode RollAutopilot shall immediately satisfy abs_roll_angle >= 30.0 => roll_hold_reference = 30.0 * sign(roll_angle)

LM_requirements AP-008B
in hdg_hold mode RollAutopilot shall always satisfy roll_cmd = hdg_hold_mode_cmd

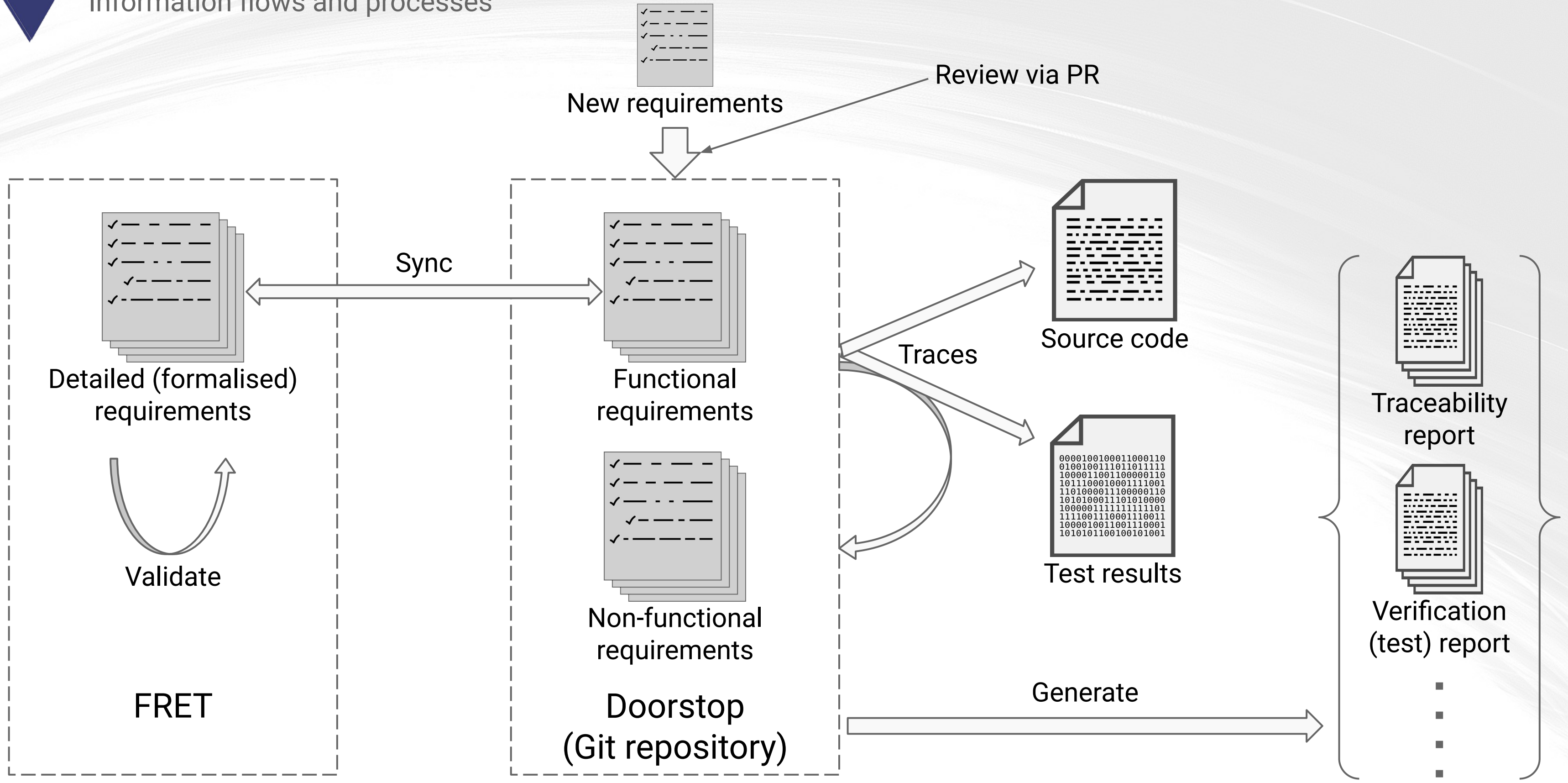
LM_requirements EUL-002B
Euler shall always satisfy R2_21 = VI_1 * R_21 + VI_2 * R_22 + VI_3 * R_23

LM_requirements REG-003
Regulator shall always satisfy count_yaw_output_exceeding_50 <= 100

LM_requirements TSM-003a
TriplexSignalMonitor shall always satisfy FC =1 => set_val =

Management of requirements in Space ROS

Information flows and processes



Management of requirements in Space ROS

Key points

- Doorstop used for:
  - High-level requirements
  - Non-functional requirements
  - Requirements traceability management
  - Artefact generation (e.g. traceability reports)
- FRET used for detailed functional requirements and consistency checks
- Requirements stored in Git (source of truth)
  - Pull requests provide a chance for requirements review
- Trace to implementation and tests via Git commit hashes

Static Analysis

Meeting the needs of aerospace with open-source analysers

- Increase code quality, ease verification
- Space ROS provides a suite of static analyzers, including IKOS and Cobra from NASA
- Currently adding dynamic analysis: code coverage and MC/DC testing
- The static analysis tools generate SARIF output
  - Most by parsing output of the tool
  - Tools should eventually support SARIF directly; would allow for more detailed information in SARIF, such as logical location
- Filtering pass to remove (some) redundancy
  - Currently, removing identical issues
  - Would like to remove semantic equivalents
- The results are made available to the Space ROS Dashboard
  - An archive format that contains analyzer output, filtered output, and metadata

IKOS (Inference Kernel for Open Static Analyzers)

Application of formal methods to support certification

- The RTCA standard DO-178C includes an extension, DO-333, that describes how developers can use static analysis in certification
    - DO-333 provides guidance on how formal (mathematical) methods may be used for the purpose of producing verification evidence suitable for use in certification
    - DO-333 lists Abstract Interpretation as a possibility to obtain certification credits
- IKOS is a static analysis framework, based on the Theory of Abstract Interpretation
    - Used to develop static analyses that are both precise and scalable
    - Makes it accessible to a larger class of static analysis developers
    - Separates concerns such as code parsing, model development, abstract domain management, results management, and analysis strategy
- References
    - https://jorgenavas.github.io/papers/ikos-sefm14.pdf
    - https://github.com/NASA-SW-VnV/ikos

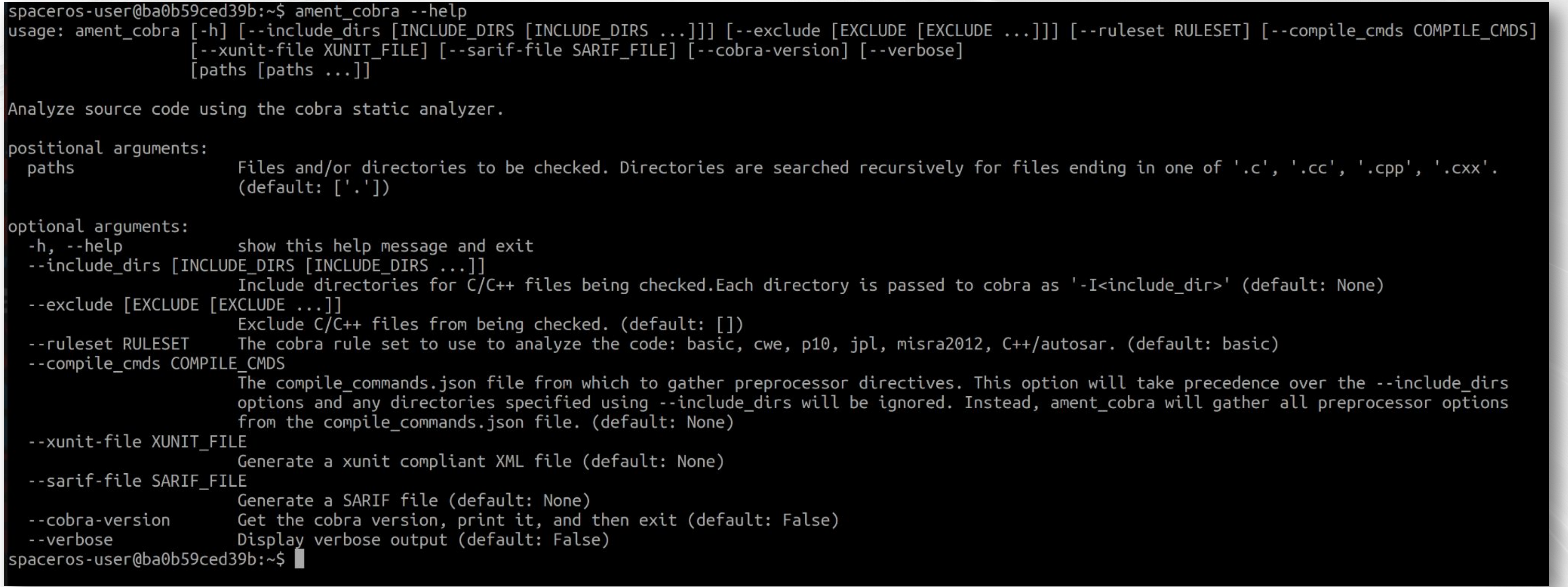
Cobra (code browser and analyzer)

An extensible, interactive tool for the analysis of C/C++ code

- A static analysis capability that works well for large code bases
- Fast analysis of general code patterns, common coding flaws, or coding rule compliance
  - Performs lexical analysis to generate a stream of language-level tokens
  - Stores the key information of source code in an extremely simple data structure
- Can be used in one of three modes
  - As an **interactive query engine** to match patterns with a simple query language
  - Execute **inline Cobra programs** that can contain arbitrary branching and iteration over the token stream to identify more complex types of patterns
  - As an infrastructure for building more elaborate **standalone checkers** that are compiled separately and linked with the Cobra code that builds the central data structure
- References
  - https://software.nasa.gov/software/NPO-50050-1
  - https://github.com/nimble-code/Cobra

Cobra (code browser and analyzer)

An extensible, interactive tool for the analysis of C/C++ code

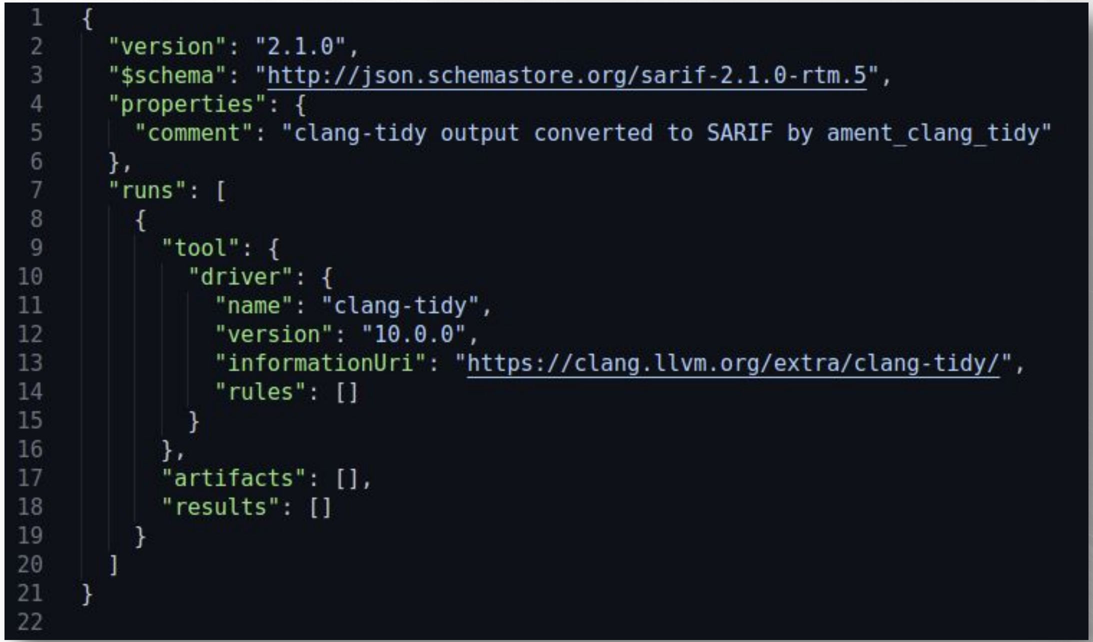


```
spaceros-user@ba0b59ced39b:~$ ament_cobra --help
usage: ament_cobra [-h] [--include_dirs [INCLUDE_DIRS [INCLUDE_DIRS ...]]] [--exclude [EXCLUDE [EXCLUDE ...]]] [--ruleset RULESET] [--compile_cmds COMPILE_CMDS]
                   [--xunit-file XUNIT_FILE] [--sarif-file SARIF_FILE] [--cobra-version] [--verbose]
                   [paths [paths ...]]

Analyze source code using the cobra static analyzer.

positional arguments:
  paths                 Files and/or directories to be checked. Directories are searched recursively for files ending in one of '.c', '.cc', '.cpp', '.cxx'.
                        (default: ['.'])

optional arguments:
  -h, --help            show this help message and exit
  --include_dirs [INCLUDE_DIRS [INCLUDE_DIRS ...]]
                        Include directories for C/C++ files being checked.Each directory is passed to cobra as '-I<include_dir>' (default: None)
  --exclude [EXCLUDE [EXCLUDE ...]]
                        Exclude C/C++ files from being checked. (default: [])
  --ruleset RULESET     The cobra rule set to use to analyze the code: basic, cwe, p10, jpl, misra2012, C++/autosar. (default: basic)
  --compile_cmds COMPILE_CMDS
                        The compile_commands.json file from which to gather preprocessor directives. This option will take precedence over the --include_dirs
                        options and any directories specified using --include_dirs will be ignored. Instead, ament_cobra will gather all preprocessor options
                        from the compile_commands.json file. (default: None)
  --xunit-file XUNIT_FILE
                        Generate a xunit compliant XML file (default: None)
  --sarif-file SARIF_FILE
                        Generate a SARIF file (default: None)
  --cobra-version       Get the cobra version, print it, and then exit (default: False)
  --verbose             Display verbose output (default: False)
spaceros-user@ba0b59ced39b:~$
```

SARIF (Static Analysis Results Interchange Format)

Unification of static analysis results

- A JSON-based exchange format for the output of static analysis tool
- Used by IDEs, code analysis tools, continuous integration systems, etc.
- SARIF output by all Space ROS static analyzers

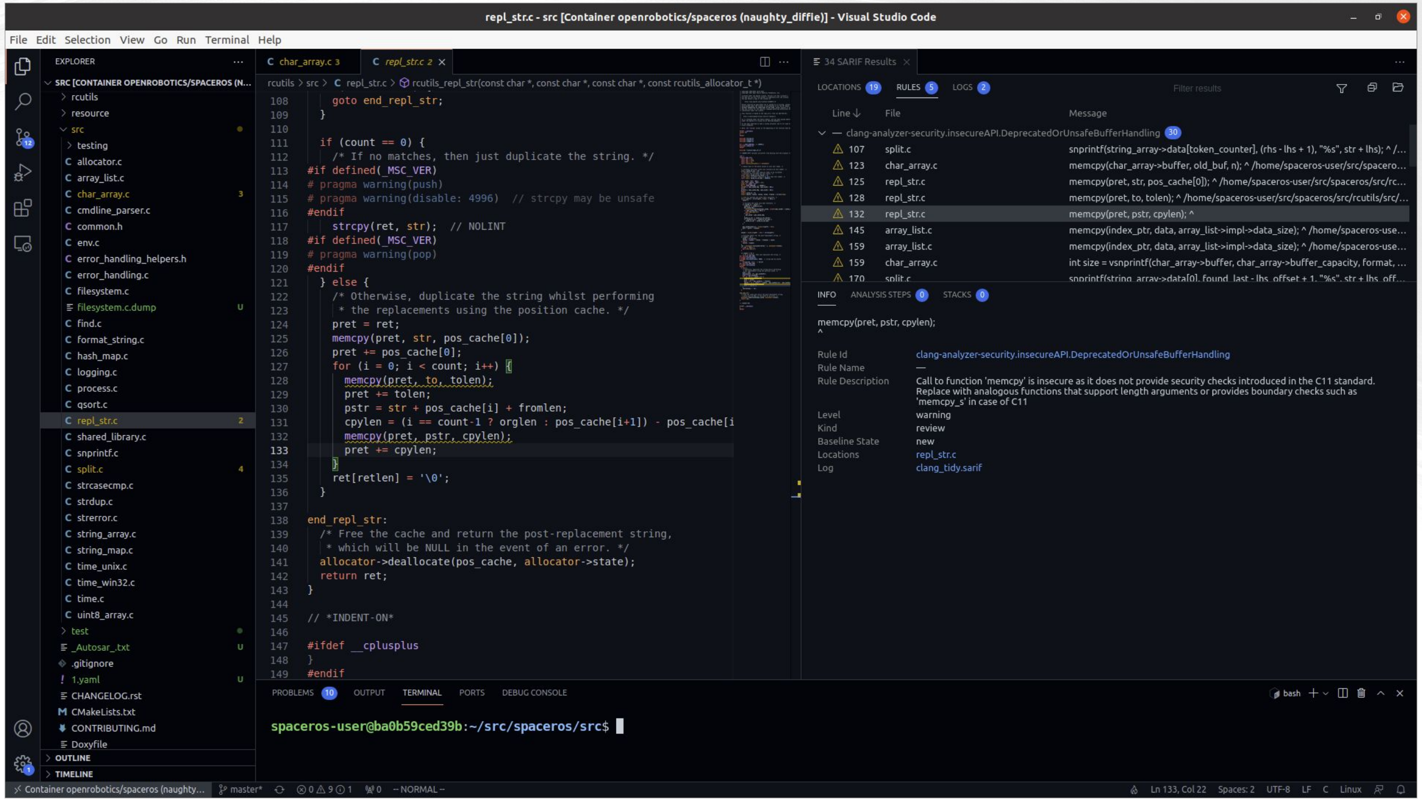


```
{
  "version": "2.1.0",
  "$schema": "http://json.schemastore.org/sarif-2.1.0-rtm.5",
  "properties": {
    "comment": "clang-tidy output converted to SARIF by ament_clang_tidy"
  },
  "runs": [
    {
      "tool": {
        "driver": {
          "name": "clang-tidy",
          "version": "10.0.0",
          "informationUri": "https://clang.llvm.org/extra/clang-tidy/",
          "rules": []
        }
      },
      "artifacts": [],
      "results": []
    }
  ]
}
```

https://docs.oasis-open.org/sarif/sarif/v2.0/sarif-v2.0.html

VSCode SARIF plugin

Making static analysis results visible



rcutils > src > repl_str.c > rcutils_repl_str(const char *, const char *, const char *, const rcutils_allocator_t *)

```c
      goto end_repl_str;
    }

    if (count == 0) {
      /* If no matches, then just duplicate the string. */
  #if defined(_MSC_VER)
  # pragma warning(push)
  # pragma warning(disable: 4996)  // strcpy may be unsafe
  #endif
      strcpy(ret, str);  // NOLINT
  #if defined(_MSC_VER)
  # pragma warning(pop)
  #endif
    } else {
      /* Otherwise, duplicate the string whilst performing
       * the replacements using the position cache. */
      pret = ret;
      memcpy(pret, str, pos_cache[0]);
      pret += pos_cache[0];
      for (i = 0; i < count; i++) {
        memcpy(pret, to, tolen);
        pret += tolen;
        pstr = str + pos_cache[i] + fromlen;
        cpylen = (i == count-1 ? orglen : pos_cache[i+1]) - pos_cache[i
        memcpy(pret, pstr, cpylen);
        pret += cpylen;
      }
      ret[retlen] = '\0';
    }

  end_repl_str:
    /* Free the cache and return the post-replacement string,
     * which will be NULL in the event of an error. */
    allocator->deallocate(pos_cache, allocator->state);
    return ret;
  }

  // *INDENT-ON*

  #ifdef __cplusplus
  }
  #endif
```

Line	File	Message
clang-analyzer-security.insecureAPI.DeprecatedOrUnsafeBufferHandling 30		
107	split.c	snprintf(string_array->data[token_counter], (rhs - lhs + 1), "%s", str + lhs); ^ /...
123	char_array.c	memcpy(char_array->buffer, old_buf, n); ^ /home/spaceros-user/src/spacero...
125	repl_str.c	memcpy(pret, str, pos_cache[0]); ^ /home/spaceros-user/src/spaceros/src/rc...
128	repl_str.c	memcpy(pret, to, tolen); ^ /home/spaceros-user/src/spaceros/src/rcutils/src/...
132	repl_str.c	memcpy(pret, pstr, cpylen); ^
145	array_list.c	memcpy(index_ptr, data, array_list->impl->data_size); ^ /home/spaceros-use...
159	array_list.c	memcpy(index_ptr, data, array_list->impl->data_size); ^ /home/spaceros-use...
159	char_array.c	int size = vsnprintf(char_array->buffer, char_array->buffer_capacity, format, ...
170	split.c	snprintf(string_array->data[0], found_last - lhs_offset + 1, "%s", str + lhs_off...

```
memcpy(pret, pstr, cpylen);
^
```

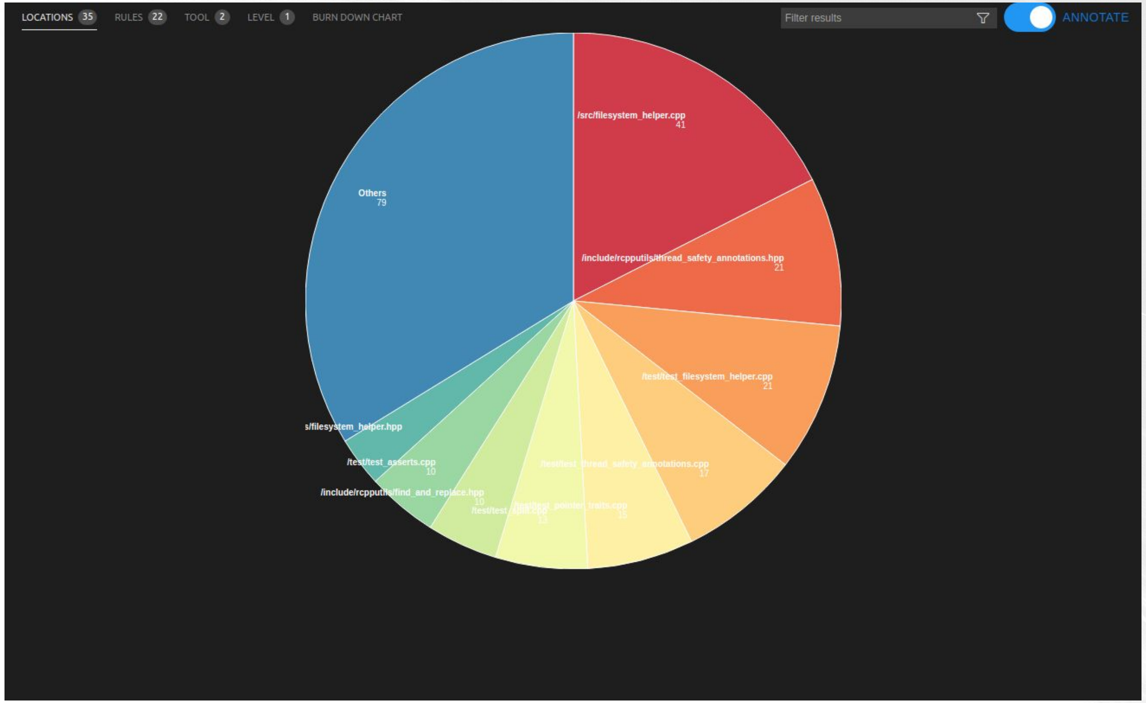
Rule Id	clang-analyzer-security.insecureAPI.DeprecatedOrUnsafeBufferHandling
Rule Name	—
Rule Description	Call to function 'memcpy' is insecure as it does not provide security checks introduced in the C11 standard. Replace with analogous functions that support length arguments or provides boundary checks such as 'memcpy_s' in case of C11
Level	warning
Kind	review
Baseline State	new
Locations	repl_str.c
Log	clang_tidy.sarif

```
spaceros-user@ba0b59ced39b:~/src/spaceros/src$ 
```

Extending the VSCode SARIF plugin

Making static analysis results visible

- Insight into static analysis, code coverage, build status, issue burndown, etc.
- A starting point for the open source community to extend and improve
- Interface to build, test, using Earthly (same as CI)
- Integrate with external dispositioning systems
- Plugin available on the VSCode Marketplace



LOCATIONS 35
RULES 22
TOOL 2
LEVEL 1
BURN DOWN CHART
Filter results
ANNOTATE
/src/filesystem_helper.cpp 41
/include/rcpputils/thread_safety_annotations.hpp 21
/test/test_filesystem_helper.cpp 21
/test/test_thread_safety_annotations.cpp 17
/test/test_pointer_traits.cpp 15
/test/test_split.cpp 13
/include/rcpputils/find_and_replace.hpp 10
/test/test_asserts.cpp 10
s/filesystem_helper.hpp
Others 79

What is Space ROS?

Space ROS 2022

Foundation

- Builds
- Releases
- Continuous Integration
- Maintenance
- Package subset
- Docker images
- Embedded target(s)

Tools and Processes

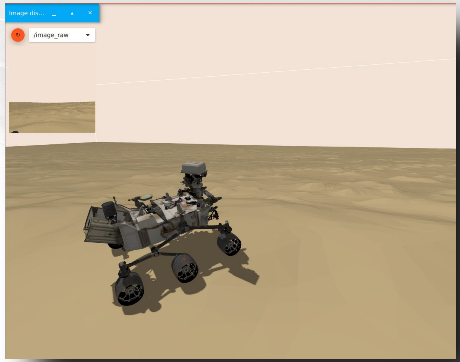
- Requirements tools and processes for traceability and analysis
- Code analysis tools with SARIF output
- Dashboard for issue navigation, visualization & dispositioning
- Development workflow
- Quality level(s)
- MC/DC testing

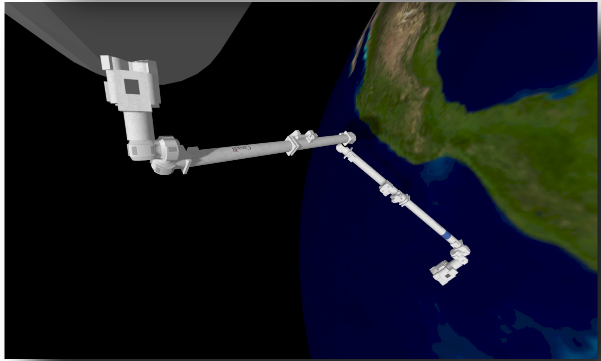
Space-Specific Functionality

- Eventing & Telemetry Subsystem
- PMR allocator
- Sample applications
- Simulation assets

Space-Specific Functionality

Item	Description
cFS/ROS 2 Bridge	TRACLabs is working on "The BRASH Integration Toolkit for ROS2 and Flight Software Interoperability."
Eventing and Telemetry Subsystem (ETS)	The Events and Telemetry System provides event reporting functionality. It is used to instrument the software that executes on the spacecraft, to retain events for later reporting, and to perform the reporting.
Custom Memory Allocators	Space ROS applications can make use of a user-supplied allocator. Provide a sample application and sample allocator that demonstrates the use of a user-supplied allocator. Provide documentation that describes how to use a user-supplied allocator.
Navigation and Manipulation Demo Apps	Incorporate navigation (Curiosity Rover) and manipulation (Candarm) demo applications.
Enable RTOS Build	Build Space ROS for the RTEMS embedded operating system, running on Qemu.
Simulation Assets	Incorporate space-related simulation assets that can then be available for use by Space ROS code.





Ongoing development

Space ROS 2023+

Foundation

- Regular releases
- Space ROS website
- Space ROS documentation site

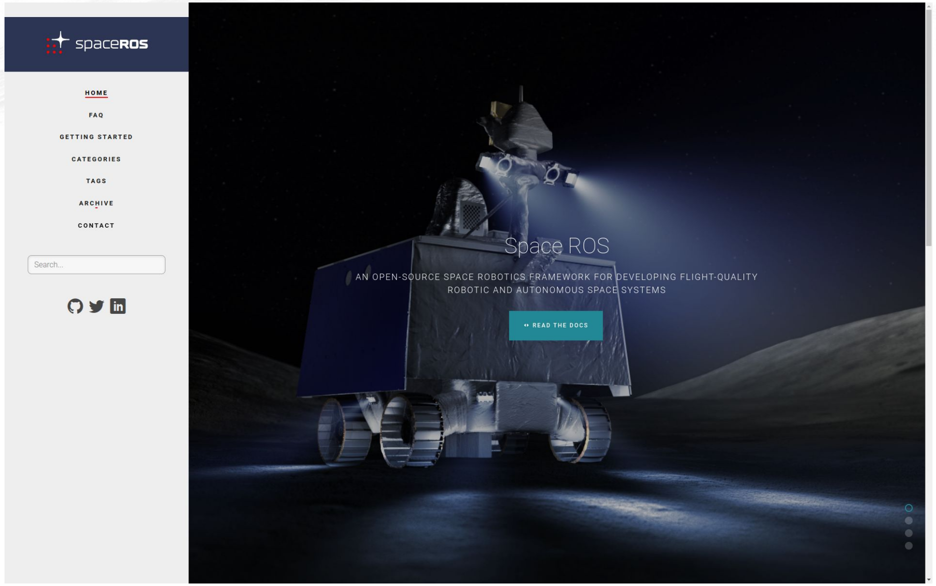
Tools and Processes

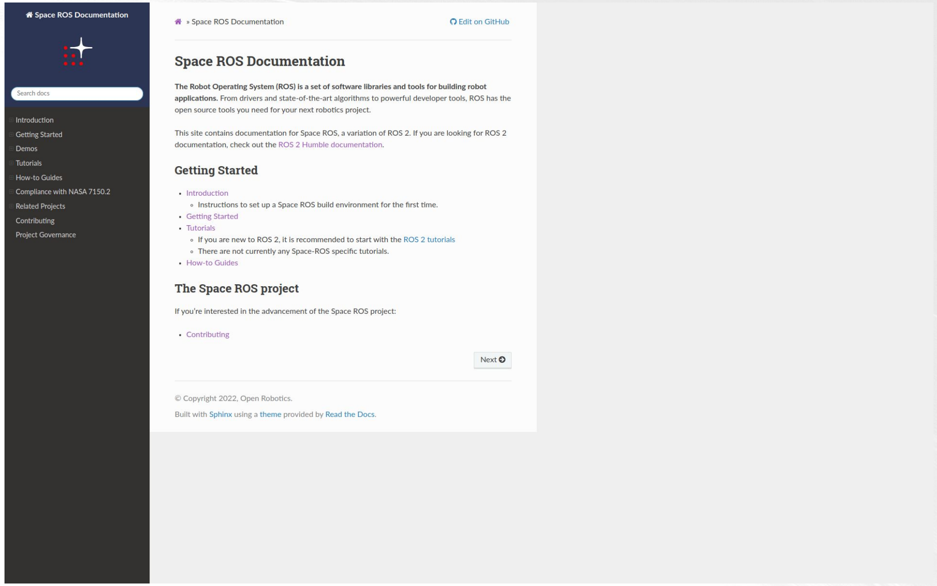
- Continue Dashboard work based on user feedback
- Add auditing support, checklists, reports
- Continue to address issues identified by code analysis tools
- Back-port requirements
- Perform requirements analysis

Space-Specific Functionality

- cFS/ROS 2 bridge
- Applications and demos

Open processes and artifacts for community-driven validation


spaceROS
HOME
FAQ
GETTING STARTED
CATEGORIES
TAGS
ARCHIVE
CONTACT
Search...
Space ROS
AN OPEN-SOURCE SPACE ROBOTICS FRAMEWORK FOR DEVELOPING FLIGHT-QUALITY ROBOTIC AND AUTONOMOUS SPACE SYSTEMS
READ THE DOCS



Space ROS Documentation

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers and state-of-the-art algorithms to powerful developer tools, ROS has the open source tools you need for your next robotics project.

This site contains documentation for Space ROS, a variation of ROS 2. If you are looking for ROS 2 documentation, check out the ROS 2 Humble documentation.

Getting Started

- Introduction
  - Instructions to set up a Space ROS build environment for the first time.
- Getting Started
- Tutorials
  - If you are new to ROS 2, it is recommended to start with the ROS 2 tutorials
  - There are not currently any Space-ROS specific tutorials.
- How-to Guides

The Space ROS project

If you're interested in the advancement of the Space ROS project:

- Contributing

Next

Open processes and artifacts for community-driven validation

- We're integrating open source tools and processes to help improve software quality
  - Requirements, code analysis, developer workflow, quality levels
  - This is done in the context of Space ROS, but could be useful to other domains
- We welcome your contributions and input
- https://github.com/space-ros

spaceROS