Project Proposal Summary Sheet

Project Number	P_2023_04
Project name	*ASAM Quality Checker Framework (qc4openx)*
Domain	Software Development
Impacted standard(s)	ASAM OpenSCENARIO, ASAM OpenDRIVE, ASAM OTX
Project type	Implementation
Start date	15.01.2024
End date	30.10.2024
TSC Submission:	30.11.2023
Proposer(s)	*Andreas Kern, CARIAD*
ASAM Office Responsible (OR)	Diego Sánchez
Initiating Companies	*(at least 3 companies are required for a valid proposal submission)* CARIAD, TrianGraphics, Porsche Engineering Services
ASAM funds	*100.000*
Backwards Compatibility	*N/A*

For more information on the ASAM project process and the proposal phase in particular, please refer to the ASAM Project Guide.

Table of Contents

1 Executive Summary

ASAM is recognized globally for its commitment to advancing standardization in the automotive sector.

ASAM standards significantly improve the exchange of data between stakeholders, but there are still various interoperability issues between users of ASAM standards. These issues often arise due to ambiguities in the standards or due to differing levels of implementation across tools.

ASAM is continually working on improving the quality of its standards, such as removing ambiguity and enriching them with examples and usage guidelines. This proposal recommends taking an additional step. It proposes the development of a comprehensive “Quality Checker Framework”. This framework will allow users to verify the conformity of files and implementations against ASAM standards, fostering greater adoption and understanding of the standards as well as significantly improving interoperability. The framework shall be standard agnostic, allowing the execution of a wide variety of both ASAM and user-defined checks for different standards. The initial development shall be based on the existing QC4OpenX Framework (Quality Checker for OpenX Standards) and shall include basic tests for ASAM OTX, which goes beyond the OpenX Standards. The existing framework has a focus on XML based input formats. The long-term strategy is to support also other input formats. This is not part of this project.

Developers shall be able to write their own rule sets that go beyond the verification of the formal correctness by means of the XML Schema files. The formal correctness is only a first step and not sufficient to assess the quality. Hence more sophisticated quality checks are required. The framework is designed to support a variety of use cases that require different types of checkers. Most applications require checkers that evaluate geometries, other physical properties or the data integrity of the static and dynamic parts of a virtual environment.

Other checkers may be created for statistical evaluation of the input files. This allows users to ensure the existence of specific objects, such as obstacles or traffic signs that are crucial for their use-case in the input files.

Additionally, there may be more checks necessary to ensure that the files can be used in a specific simulation environment, because not all simulation environments support the full feature set of the standard.

Besides performing static files checks, which read, evaluate and analyze the content, it can also be feasible to create checkers that run a simulation and analyze the simulation results, since non-plausible simulation behavior may point to problems in the input files.

A base suite of checks for standards shall be directly developed, maintained and hosted by ASAM; these shall include syntax and semantic checks that directly reduce ambiguity in the standards. This proposal includes the definition and implementation of a first series of base checks for a first set of standards: ASAM OpenDRIVE, ASAM OpenSCENARIO, and ASAM OTX. These shall be packaged as “checker rules”, to be published and continually developed alongside future revisions of the standards.

The framework shall also enable the use of additional user-defined checks, whether proprietary or open, to meet the requirements of the wide spectrum of users of ASAM standards. A detailed configuration management and result reporting shall also be included.

Project participants in this project shall detail the requirements towards a checker framework and define the checker rules for the first set of standards named above. A service provider, contracted by ASAM, shall implement the framework and the corresponding checker rules.

The outcome of this project shall be a framework published by ASAM, as well as checker rules and executable checkers for the aforementioned standards. All implementations, the framework and the executable checkers, shall be open-source and licensed under MPL-2.

2 Overview / Goals

2.1 Motivation

Standardized formats play a crucial role in promoting efficiency, interoperability, and consistency across various industries and domains. They simplify data exchange, reduce development efforts, and contribute to cost savings while ensuring that data remains accessible, compatible, and secure.

As adoption of standards in the industry continues to grow, there is an ever-increasing need to ensure consistent usage and homogeneous implementation. Standardized data formats are often interpreted differently, leading to differences in implementations. These differences in interpretation can lead to compatibility issues and hinder interoperability.

Ambiguities in the Standard: Some standardized formats may have ambiguities, vague language, or room for interpretation within the standard itself. This can result in different implementations that conform to the standard but have varying interpretations of certain elements.
Lack of Strict Compliance: Implementers may not strictly adhere to all aspects of the standard, opting to prioritize other considerations such as performance or compatibility with legacy systems. This can result in deviations from the standard.
Interactions with Other Standards: In complex systems, data may need to conform to multiple standards simultaneously. Conflicts or inconsistencies between these standards can lead to variations in how data is interpreted.
Platform or Language Differences: Different programming languages, libraries, or platforms may be used to implement the same standard. These variations can affect how data is encoded, parsed, and processed.
Human Error: Human errors during the implementation or configuration of systems can lead to discrepancies in how data is handled and interpreted.

Standards are key to enabling exchange of information between stakeholders. Having an additional mechanism to ensure consistency of this information during exchange will strengthen the adoption rate of ASAM's standards and assist the market in creating large, shared pools of standard-compliant data assets.

2.2 Relations to Other Standards, Projects, or Organizations

2.2.1 Standard and Standardization activities

It is expected that the deliverables of this project include discrete checker rules for the following standards:

- ASAM OpenSCENARIO
- ASAM OpenDRIVE

These shall be integrated into the next releases of these standards as normative checker rules. See 3.2.4 Rule Integration Process for further detail.

To demonstrate the applicability of the framework for further standards, the rules defined for the Base Standard (BS) for ASAM OTX 3.2.0 shall also be implemented.

3 Technical Content

3.1 Checker framework

3.1.1 High level requirements

Environment Flexibility: The framework will be designed to function seamlessly both in local machine environments and server-based settings, interactive usage or run in automated workflows allowing users flexibility in deployment.

Traceability: One of the primary goals is to provide clear traceability of issues to source code and into a graphical view of the virtual environment. This will enable users to swiftly identify and rectify the discrepancies in the data or implementation.

Certification: The framework shall include common ASAM rule sets, which define the common understanding of the standard to ensure the portability of files.

Extensibility: Recognizing the diverse technological ecosystems, the framework will permit defining checks in any programming language. This ensures the widest possible participation and utilization.

Parameterization: The checks can be parameterized, enabling users to adjust the verification process's stringency or focus according to their specific needs.

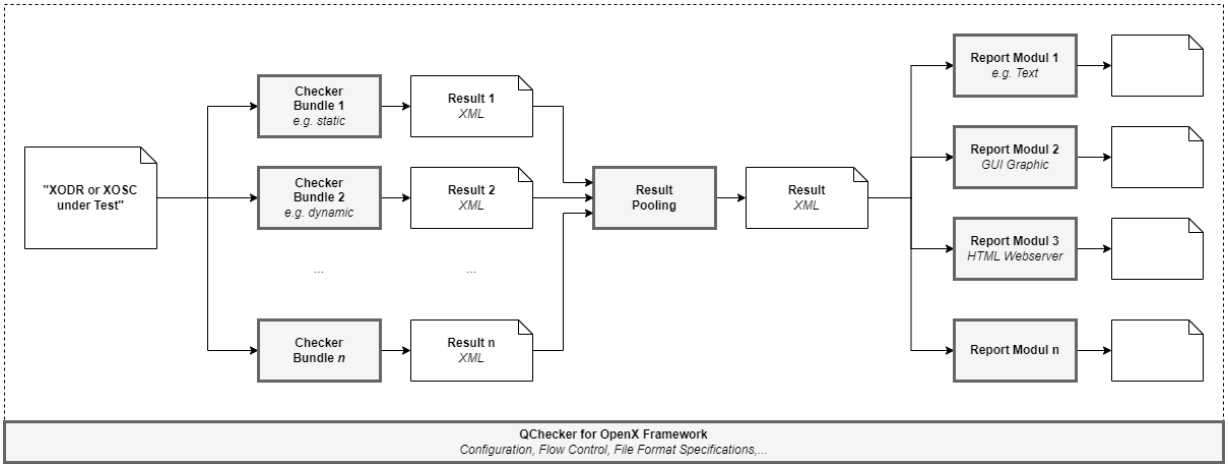
Static and dynamic checks: Besides classic static file checks, it shall be possible to run the input files in a test system and create checker results based on the analysis of the simulation run.

Reporting Capabilities: The framework will provide detailed reports in diverse machine and human readable formats. A dedicated GUI will allow users to interact with the results, offering features like filtering by rule sets and directly viewing discrepancies in the input file source code.

3.1.2 Architecture/Workflow

To cater to the complex and layered nature of ASAM standards, the different approaches to implement checks and the different sources of implementations, the framework will allow the bundling of checks into executable "checker bundles". The results of the individual executables will be merged into one result file. This can be converted into different report formats or used for interactive analysis of the found issues.

This modular approach ensures comprehensive coverage and easier management.



The framework itself delivers besides the checker bundles for common checks:

- File format specifications for the configuration and the machine-readable result files.
- Configuration GUI for the selection and parametrization of checker bundles and their checks.
- Runtime to execute a configuration of the framework.
- Standard report modules for creating a human readable text report and for interactive analysis of the checker results in the source files.
- 3D viewer to show issues in the virtual environment (OpenDRIVE map). The geometry generation shall be based on existing solutions (e.g. libOpenDRIVE or the “odrviewer of esmini).

3.1.3 Configuration and runtime

The existing Qt based configuration GUI is available on Windows and is able to trigger existing checker bundles to write an empty report file containing information about included checks and their parametrization possibilities. This approach is used for creating default configuration, which can be adapted. The GUI also includes the runtime to trigger the individual modules of a configuration.

The GUI shall also be runnable on Linux. All other existing modules are already usable on Linux.

The runtime to trigger the configured executables shall be separated from the GUI for easier maintenance and development of new features in the future.

The GUI and runtime shall be able to include and run checker bundle and report module executables from other directories than the main build of the framework. The current implementation needs all executables in the same directory.

3.1.4 Results and reporting

The current implementation of the result pooling and the report module for generating a human readable text report needs no new additional features for the project.

The interactive report GUI has a size limitation for the input files of 5 MB. Bigger input files are not shown. So, traceability into the source code is only available for smaller files. This comes due to limitations of the used Qt component for displaying the source code including the XML

syntax highlighting. This limit shall be removed by using another component or by optimizing the current implementation.

The current implementation has a plugin interface to connect to a 3D visualization and show issues in the virtual environment (generated geometry of the used OpenDRIVE file). The project shall use an existing implementation for the 3D visualization to show issues in the map in the interactive mode.

3.1.5 Examples, testing and documentation concept

The current architecture and implementation provide the possibility that checker bundles can be queried by the configuration GUI about their available checks and their parametrization possibilities. Additionally, it shall be possible that the checker bundles also can be queried about the documentation (e.g. Markdown based) of the implemented checks. The idea is to get a self-documented framework for the current configuration.

The project shall also provide example input files for all implemented rules to demonstrate the capabilities of the delivered checker bundles. These input files shall also be used for the functional testing of the implementation of the checker bundles. Together with the described self-documented checker bundle implementation, it is possible to provide comprehensive documentation of the whole framework delivery.

3.1.6 Library for implementing own modules

The current implementation provides a C++ base library for reading and writing the configuration format and the machine-readable result file. Due to the heavy usage of Python in the domain, the project shall implement and provide a Python library for reading and writing the configuration format and the machine-readable result file.

3.1.7 Implementation

The project group shall define requirements towards the framework. Implementation shall be done by a service provider, contracted by ASAM.

3.2 Checks

3.2.1 Framework

The framework for implementing checks includes the following points.

Structure: Depending on the file to be tested, format-specific tests are run. These are organized thematically by categories (schema, semantics, geometry, tool compatibility, linkage and statistics) and can contain a variety of check scripts. Issues found can be registered by specifying the error level, description and location. The registered issues can be written as text, JSON and xqar files.

Environment flexibility: The framework for the checks is implemented in Python, enabling local and server-side execution and can also be integrated into automatic processes.

Expandability: Expanding checks for new formats, your own categories and individual checks is done by adding new folder structures and check scripts. The checks are automatically

registered and called when executed. Additional folders with checks can also be specified when running.

Configurability: The execution order of categories and their checks can be specified to execute relevant checks such as schema check first. Flexible variables can be configured individually for each check.

Integration into Checker Framework: The framework for the checks creates an xqar result file, which can be processed or viewed in the Checker Framework. Jump points (locations) in the file, XML path and street position are supported.

Traceability: An example is included for each check script to demonstrate the issue(s) of the check.

Complexity: The framework for the checks currently contains 30 checks with 60 issue reports for ASAM OpenSCENARIO and ASAM OpenDRIVE. In addition to schema checks, many semantic and geometric checks are integrated for ASAM OpenDRIVE, and a dynamic tool check based on esmini for ASAM OpenSCENARIO.

+ Anzeige für OpenCRG und OTX.

3.2.2 Rules

The project shall define the textual rule format, documentation requirements and the process of acquiring them. Additionally, the requirements for positive and negative examples shall be defined.

The project will define an approach for structuring rule sets and which rule sets need to be aligned with which stakeholder and who will be provider for the runnable checker bundles.

A series of rules shall be defined for ASAM OpenSCENARIO and ASAM OpenDRIVE. These rules shall enable full syntactic validation of elements/objects against the respective schemata of the standards.

Additional rules for a certain subset of semantic checks shall be defined, the scope of this is to be defined within the first stages of the project. This project will not aim to define an exhaustive range of semantic checks for the standards. Rather, these shall cover some of the most frequent issues occurring in exchange of files based on the respective standards.

3.2.3 Check implementation

The rules defined by this project shall be implemented by a supporting service provider in a consistent format & technology, agreed on during the contracting process.

In addition to the implementation of checks for the rules in section 3.2.2, to demonstrate the functionality for existing rulesets in the ASAM portfolio, the ~60 rules defined for the OTX 3.2.0 base standard shall be implemented.

3.2.4 Rule integration process

On completion of this project, the definition of further rules, whether for the standards already addressed here or others, shall take place solely within the scope of development projects for

the respective standards. The set of rules for ASAM OpenDRIVE and ASAM OpenSCENARIO 1.x defined in this project shall be integrated into the next release of the respective standards. Any future development projects for these standards shall define additional checks for any new or modified features in new releases. A process for this shall be outlined within this project.

3.3 CI/CD

The framework and the base ASAM rule sets shall be built in the GitHub pipeline to provide a ready to use executable for users. This process should be triggered automatically on code changes and usable for all development branches.

Additionally, the framework and checks shall be compatible with CI/CD workflows.

4 Deliverables

At the end of the project, the project group will hand over the following deliverables to ASAM:

Table 1 Deliverables

Item No.	Description
1	QA framework implementation (source code, hosted on ASAM Github)
2	QA Framework documentation
3	Checker rules & checker library: ASAM OpenSCENARIO XML up to 1.3.0
4	Checker rules & checker library: ASAM OpenDRIVE up to 1.8.0
5	Checker library: ASAM OTX 3.2.0 – Base standard only
6	
7	

4.1 Review Process

The process for deliverable review documented in the project guide is applicable to all projects (see here).

The ASAM OR will provide further details on quality criteria and tools used prior to the initiation of a review in a project.

This project shall perform a full **public review.**

5 References

Provide a list of documents and their authors that are referenced in earlier chapters. Use the sequential number in squared brackets for referencing them in earlier chapters.

See guidance on References in the ASAM Editorial Guide

[1] <Author 1 Last Name>, <Author 1 First Name>; <Author 2 Last Name>, <Author 2 First Name>; <…more Authors…>: <Title>; <Publishing House>; <Year>; ISBN: <ISBN-13 Number>

[2]