ASAM Trace Data Format 2.0.0

Release Presentation

18 Nov, 2025

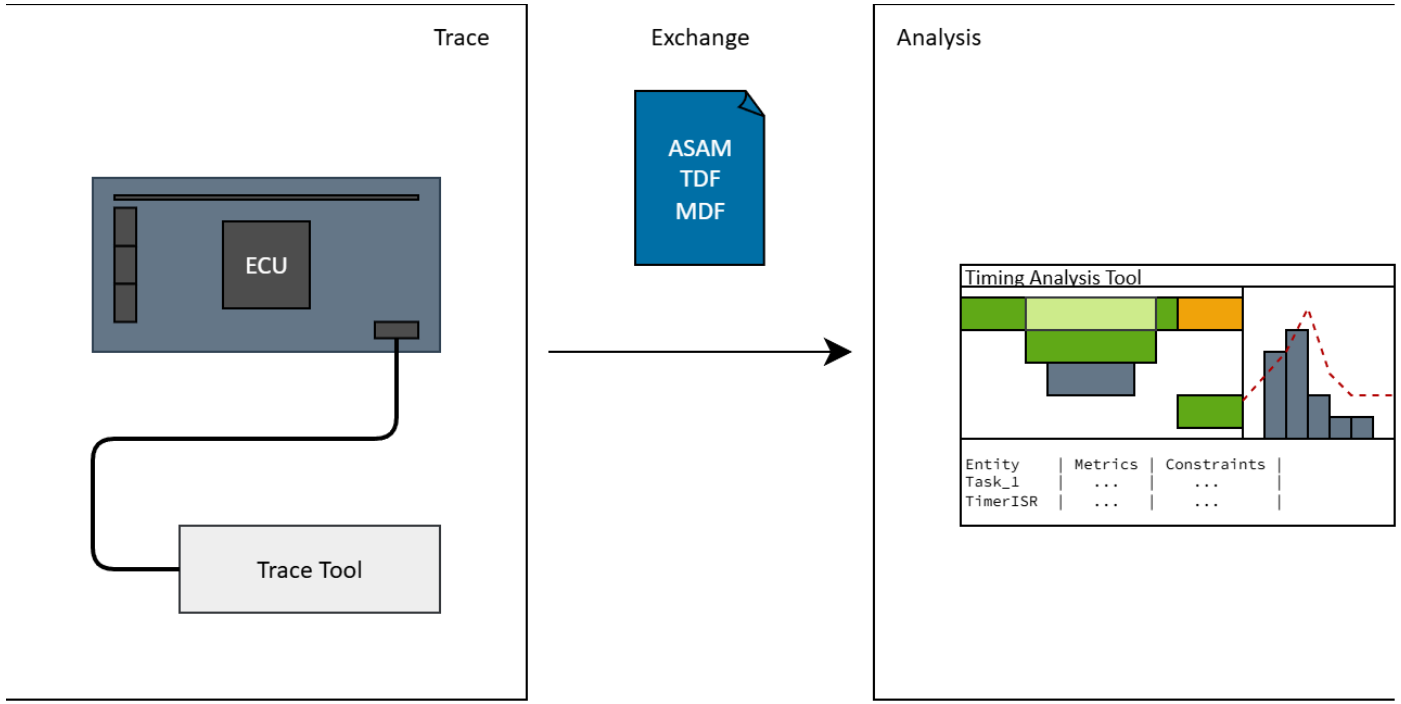
ASAM Association for Standardization of Automation and Measuring Systems

Agenda

1	Introduction
2	Motivation for New Release
3	New Features
4	Other Changes
5	Backward-Compatibility
6	Relation to Other Standards
7	Deliverables

Introduction

- ASAM TDF provides a standardized exchange format for trace measurements of scheduled run-time entities such as functions, threads, interrupts or similar items.
- It is a common interface between trace and analysis tool vendors to connect the low-level real-time tracing of the software with higher level timing analysis.



Introduction

- The format supports the recording of system events and execution traces in a generic, system-agnostic way, enabling broad applicability across different embedded operating systems.
- It also supports storing timing and performance metrics alongside the trace data.
- It uses the ASAM MDF file format as a container to enable efficient integration with tools for writing, analysis, visualization, and validation.
- It further uses features provided by MDF to present the stored data in a self-documenting and human-readable way without sacrificing the performance advantages of a binary file.
- External descriptions (for example ARXML files) can be referenced and linked to the TDF file.

Motivation

- ASAM TDF 2.0.0 is a new version of ASAM Run-Time Interface 1.0.0
- It was originally developed with a focus on compatibility with the AUTOSAR Run-Time Interface standard, but the scope of TDF has since been broadened to make it suitable for use in both AUTOSAR and non-AUTOSAR contexts.
- The AUTOSAR ARTI standard has also evolved and no longer matches details described in ASAM TDF.
- Using ASAM ARTI V1.0.0 in real project revealed inconsistencies and inconveniences we want to address with this new version.

New features

Overview

- New Structure for Trace Classes
- Uniform Handling of the Core Identifier
- Inclusion of an Event Counter to Mitigate Data Loss
- Addition of a Trace Class for Processes
- Addition of a Trace Class for Virtual Machines
- Split Trace Class Definitions and AUTOSAR Mapping

New features

New Structure for Trace Classes – Changes

- A “Trace Class” describes the type of the traced entity.
  There are several different trace classes because different types of entities are associated with different states and transitions between them.
- The MDF channel structures to store these trace classes has changed with version 2.0.0
- The new structure is derived from the type of entity and the information that is important for it.
- The new structure avoids references to AUTOSAR terminology whenever possible to better accommodate non-AUTOSAR applications.

New features

New Structure for Trace Classes – Example for Version 1.0.0

- Version 1.0.0 defined the trace classes AR_CP_OS_TASK and AR_CP_OSARTI_TASK.
- These names directly tie them to AUTOSAR (AR) Classic Platform (CP) Tasks.
- The exact meaning of OS_TASK and OSARTI_TASK is also specified in AUTOSAR which narrows the scope of these trace classes even further.
- The MDF channel structure of these classes consisted of 4 channels
  - instanceName – Virtual channel for OS name
  - instanceParameter – Channel for Core ID information
  - eventName – Channel for trace event based on trace class
  - eventParameter – Channel for trace object id

New features

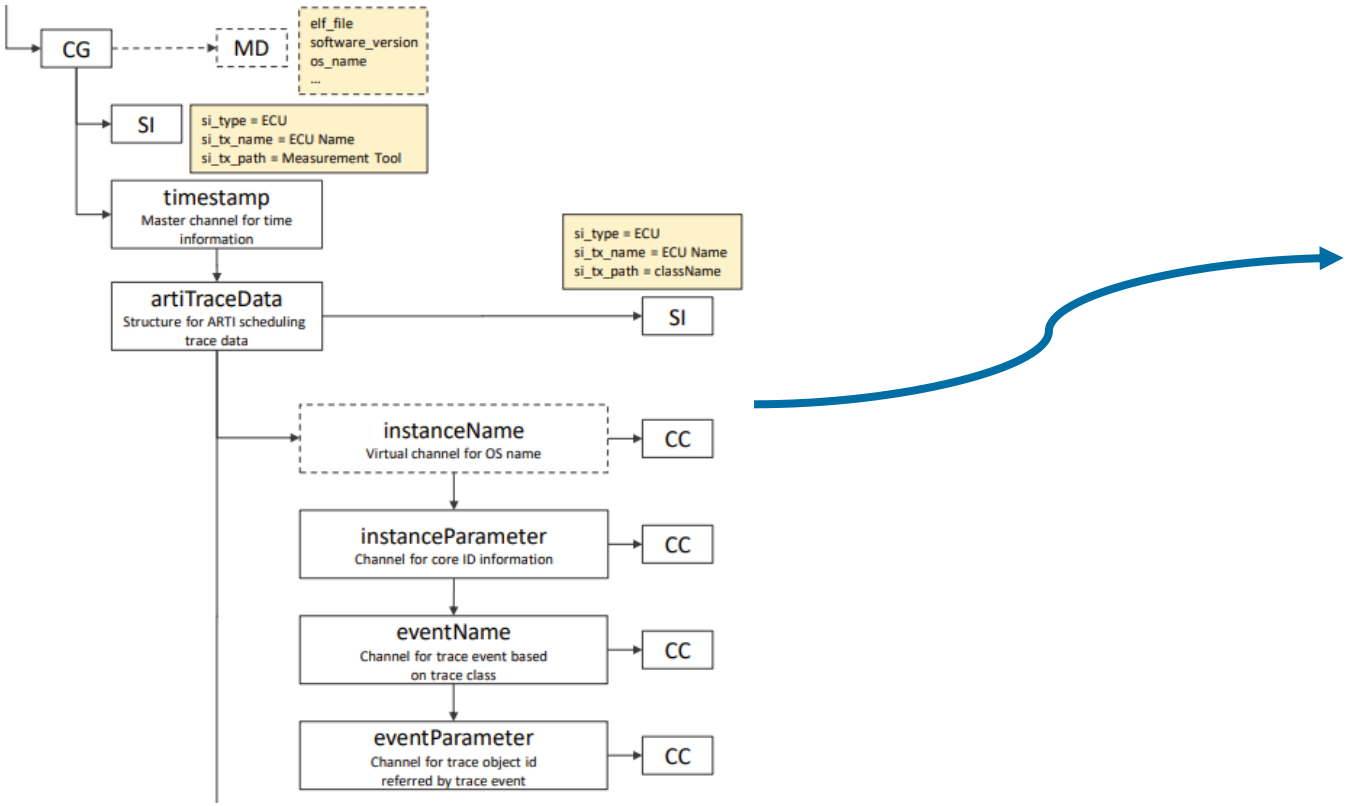
New Structure for Trace Classes – Example for Version 2.0.0

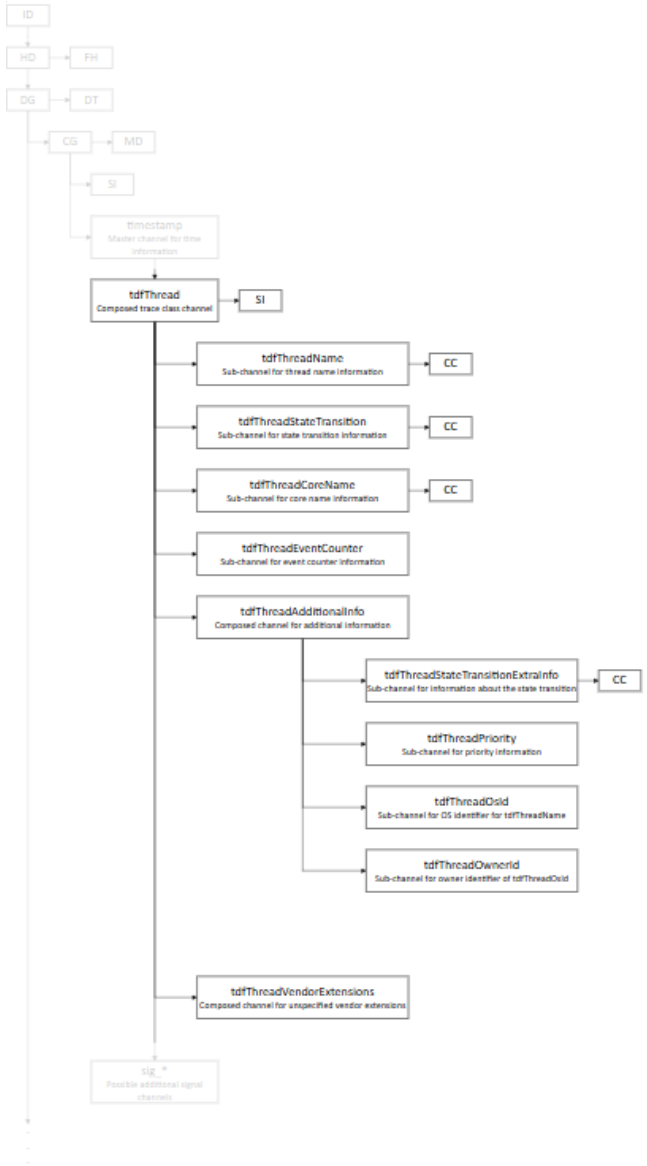
- Version 2.0.0 consolidated and generalized these two trace classes into one.
- The new trace class tdfThread can represent both of these AUTOSAR Classic Platform Task types as well as AUTOSAR Adaptive Platform Threads and non-AUTOSAR entities.
- The channel structure of tdfThread consists of
  - tdfThreadName
  - tdfThreadStateTransition
  - tdfThreadCoreName*
  - tdfThreadEventCounter*
  - tdfThreadAdditionalInfo
    - tdfThreadStateTransitionExtraInfo
    - tdfThreadPriority
    - tdfThreadOsId
    - tdfThreadOwnerId
  - tdfThreadVendorExtensions

* More details on slides 13 & 15

New features

New Structure for Trace Classes – Example Version 1.0.0 vs. 2.0.0





New features

Uniform Handling of the Core Identifier – Changes

- The MDF channel structure being tied directly to the ARTI_TRACE(…) macro has led to some inconsistencies regarding the handling of the core identifier.
- This piece of information has been identified to be mandatory for every trace event.
- Version 2.0.0 addresses this by making the corresponding MDF channel mandatory
- Details that led to the inconsistencies have also been clarified.

New features

Uniform Handling of the Core Identifier – Example

Table 3-9: Trace Class USER_STOPWATCH

Parameter	Type	Description
className	Token, literal text	USER_STOPWATCH
instanceName	Token, literal text	Value that identifies the instance of the stopwatch.
instanceParameter	uint32	Not used, should be set to 0.
eventName	Token, literal text	Identifies a state transition event. One of the following literals: • Start • Stop
eventParameter	uint32	Not used, should be set to 0.

Table 36. Trace Class tdfStopwatch

Sub-Channel name	Description	Required
tdfStopwatchName	A human-readable identifier for the entity undergoing a state change, which shall be unique within this trace class.	Yes
tdfStopwatchStateTransition	A human-readable identifier of the state transition event. Shall be one of the following: • Start • Stop	Yes
tdfStopwatchCoreName	A human-readable identifier of the core on which the event occurred on.	Yes
tdfStopwatchEventCounter	A counter containing the number of events observed on the core identified by tdfStopwatchCoreName since the start of the trace recording.	No

New features

Inclusion of an Event Counter to mitigate data loss – Changes

- The MDF file stores data of different trace classes in different channel groups.
- This can lead to ambiguity when reading the file in case of timestamp collisions.
- Timestamp collisions are expected due to the short timespans involved when recording schedule traces.
- It would be sufficient to be able to restore the order in which events have originally been seen by the trace tool.
- Version 2.0.0 addresses this by specifying an optional channel to store an additional piece of information that can be used to restore the original order of events.

New features

Addition of a Trace Class for Processes

- The trace class tdfProcess was added mainly to better accommodate AUTOSAR Adaptive Platform and non-AUTOSAR system (for example Linux).
- It also maps to the AUTOSAR trace class AR_CP_OS_APPLICATION.
- All trace classes have been generalized like this but feature parity with the 1.0.0 version has always been considered.
- The trace class tdfStopwatch for example was added mainly to still fully support the AUTOSAR ARTI standard.

New features

Addition of a Trace Class for Virtual Machines

- The tracing of a hypervisor and multiple virtual machines is a use case that is not covered by version 1.0.0.
- Feedback from the project members indicates that this use case has risen in importance recently.
- Version 2.0.0 added the trace class tdfVirtualMachine because the presence of multiple machines on shared hardware has to be considered when evaluating the timing characteristics of the system.

New features

Split Trace Class Definitions and AUTOSAR Mapping

- ASAM ARTI V1.0.0 was tightly coupled to AUTOSAR ARTI Classic Platform.
- ASAM TDF V2.0.0 is decoupled from AUTOSAR to be more flexible.
  - V2 is able to serve AUTOSAR Adaptive Platform.
  - V2 is able to serve non-AUTOSAR applications.
- AUTOSAR Classic Platform continues to be an important use case.
- Thus, a standardized mapping is introduced, how to store AUTOSAR ARTI CP events in an ASAM TDF trace file.
- Each AUTOSAR trace class maps in a well-defined way to an ASAM trace class.
- This allows producers (trace tools) and consumers (timing analysis tools) of ASAM TDF to create and interpret the data consistently.
- *Example*: The AUTOSAR trace class "`AR_CP_OS_TASK`" maps to the ASAM channel "`tdfThread`" with its parameters:

AUTOSAR parameter	ASAM channel
instanceParameter	tdfThreadCoreId
eventName	tdfThreadSateTransition
eventParameter	tdfThreadName

Other changes

Storage of ARTI Timing and Performance Metrics in MDF

- The corresponding sections have been updated to work with the new trace class structure.

Definition of Timing and Performance Metrics

- The included definitions have been updated to consider the potential presence of multiple virtual machines.
- Deviations of the use case agnostic definitions are included to better accommodate AUTOSAR Class Platform environments.

Bit pattern storage of artiVariableValue (USER_DATAPOINT in 1.0.0)

- The expected bit pattern stored for channel tdfVariableValue is described more precisely.
- tdfVariableValue can now also store 64-bit values.

Backward compatibility

ASAM Trace Data Format 2.0.0 is a new version of ASAM Run-Time Interface 1.0.0

- It is not backwards compatible.
- Data encoded according to this new standard can be identified by a newly introduced element of metadata.
- Migrating an existing file is possible because ASAM Trace Data Format 2.0.0 can accommodate all use cases version 1.0.0 was capable of, but existing tools are not expected to be able to perform this migration.
- Old tools that only support 1.0.0 will not be able to find any valid data in 2.0.0 files due to the structural changes.
- It is recommended that new tools developed against version 2.0.0 check the metadata entry which contains the standard version.

Relation to other standards

ASAM Run-Time Interface 1.0.0

- ASAM Trace Data Format 2.0.0 is a new version of that standard.
- Version 2.0.0 is not backwards compatible.

ASAM MDF 4.1.1

- ASAM Trace Data Format 2.0.0 uses terms and definitions described in the MDF base standard.

AUTOSAR Run-Time Interface

- ASAM Trace Data Format 2.0.0 uses terms and definitions described by the AUTOSAR development partnership in the Specification of AUTOSAR Run-Time Interface.

IEEE 754-2019

- ASAM Trace Data Format 2.0.0 references the IEEE standard to disambiguate the encoding of floating-point values.

Deliverables

Documents

- ASAM_TDF_Specification_V2_0_0.pdf

Supplementary Files

- ExampleScenario.drawio.png
- ExampleScenario.xlsx
- ExampleScenarioImplementation1.mf4
- ExampleScenarioImplementation2.mf4

Notes

- The example scenario is included in a visual as well as a textual version.
- Two implementations of the example scenario are provided. They differ in regard to details like channel sizes which are not prescribed by the standard. Both are considered valid ways of storing the example scenario in MDF.