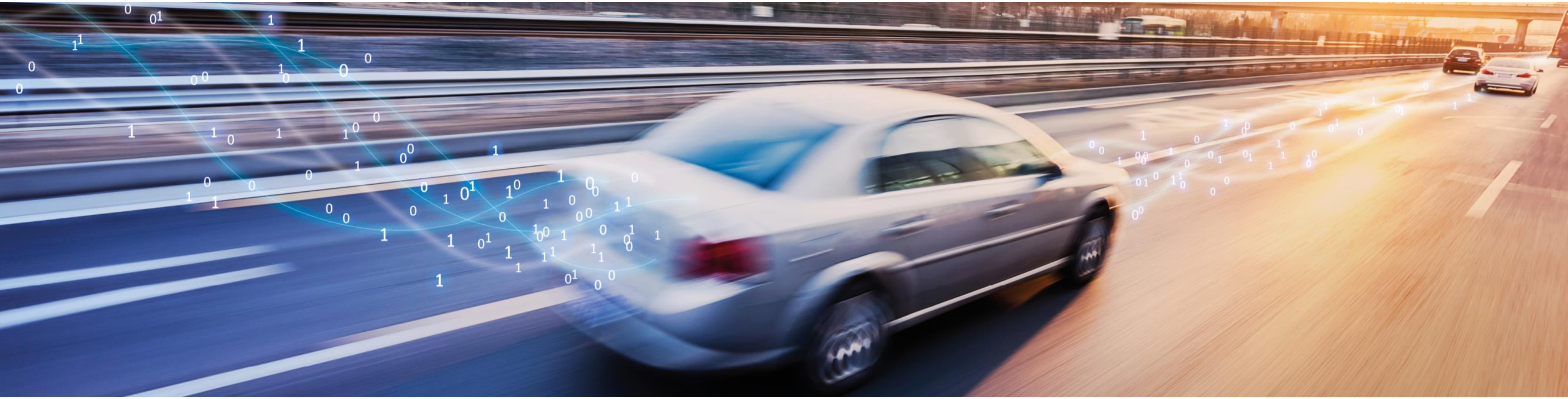
ASAM AE XIL-MA v2.1

Release Presentation

Dr. Rainer Rasche, dSPACE GmbH

2017-10-18

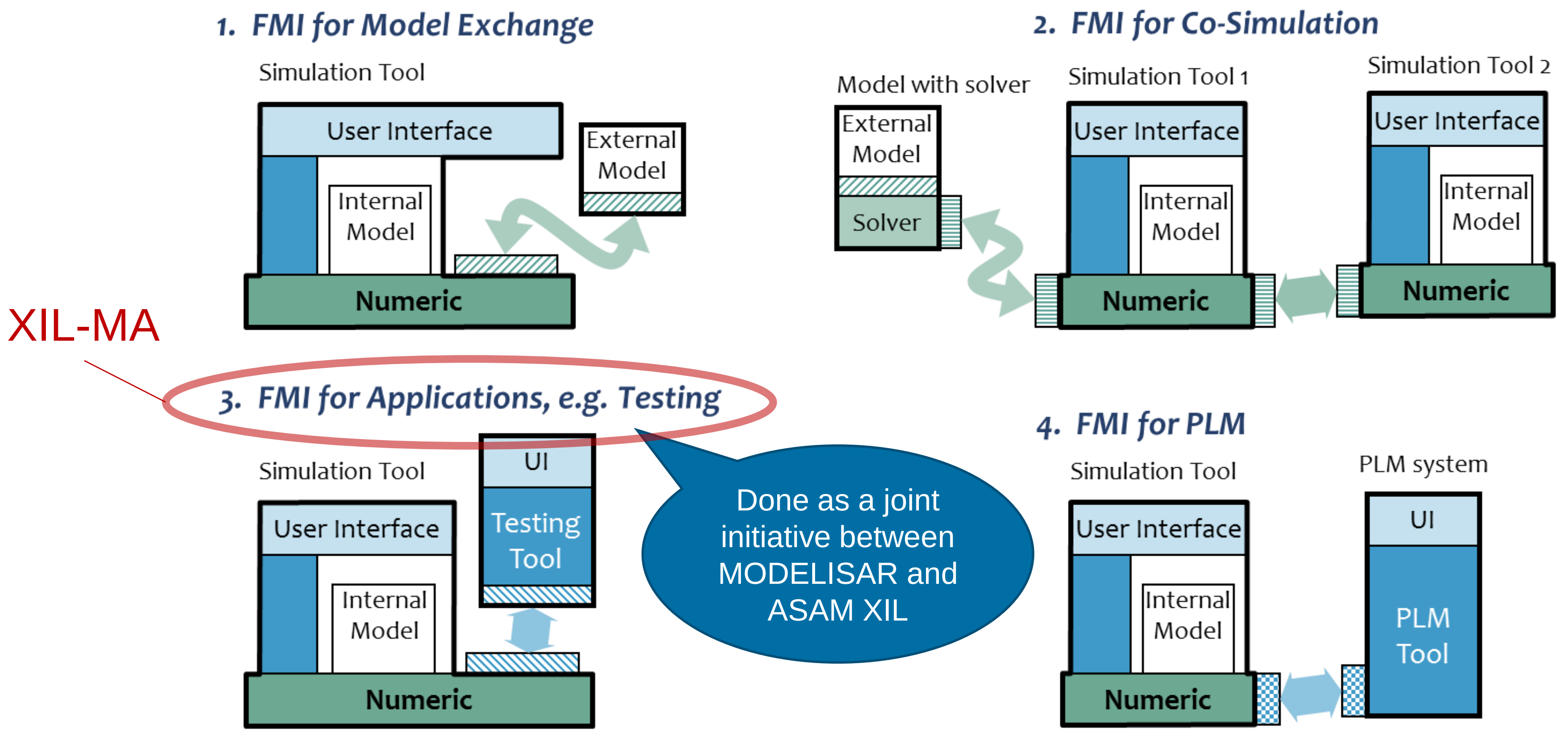


Agenda

Motivation for XIL-MA

Cooperation of ASAM XIL and ITEA 2 Project MODELISAR

European Project MODELISAR (2008 – 2011)
was setup to develop a set of open interface standards for simulators



ASAM

Background on FMI (1)

Result of MODELISAR: Functional Mockup Interface FMI

- Open standard
  - Free downloads on FMI web page: https://www.fmi-standard.org/downloads
- Specification is released under a copyleft license
  - CC BY-SA: Creative Commons Share Alike
- Code is released under open BSD license
  - BSD: Berkeley Source Distribution

FMI Version 2.0

FMI for Model Exchange and Co-Simulation

This is the second version of the Functional Mockup Interface standard (FMI). It is a major enhancement compared to FMI 1.0, where the FMI 1.0 Model Exchange and Co-Simulation standards have been merged, and many improvements have been incorporated, often due to practical experience when using the FMI 1.0 standards. New features include: Parameters can be changed during simulation, the complete FMU state can be saved, restored and serialized, directional derivatives with respect to states and inputs can be computed, the structure of the partial derivatives with respect to states and inputs can be given (to support large systems), algebraic loops over FMUs are now supported in all modes (initialization, event, continuous-time) for Model Exchange, allowing for example improved initialization.

Version 2.0 was released on July 25, 2014.

FMI Version 1.0

FMI for Model Exchange

The intention is that a modelling environment can generate C-Code of a dynamic system model that can be utilized by other modelling and simulation environments. Models are described by differential, algebraic and discrete equations with time-, state- and step-events. The models to be treated by this interface can be large for usage in offline or online simulation or can be used in embedded control systems on micro-processors. It is possible to utilize several instances of a model and to connect models hierarchically together. A model is independent of the target simulator because it does not use a simulator specific header file as in other approaches. A model is distributed in one zip-file called FMU (Functional Mock-up Unit).

Version 1.0.1 was released on July 10, 2017.
Version 1.0 was released on Jan. 26, 2010.

FMI for Co-Simulation

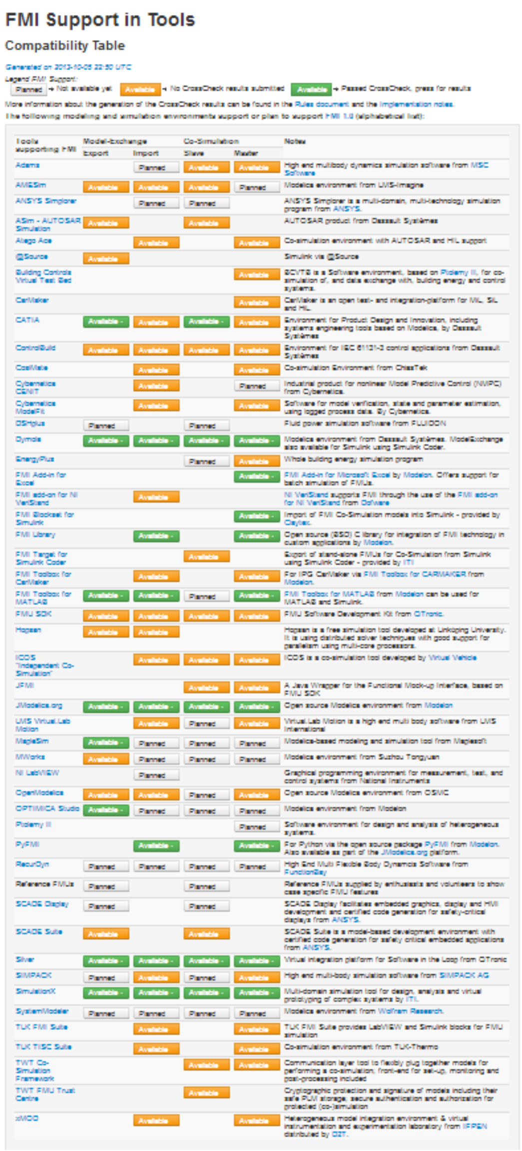
The intention is to provide an interface standard for coupling two or more simulation tools in a co-simulation environment. The data exchange between subsystems is restricted to discrete communication points. In the time between two communication points, the subsystems are solved independently from each other by their individual solver. Master algorithms control the data exchange between subsystems and the synchronization of all slave simulation solvers (slaves). All information about the slaves, which is relevant for the communication in the co-simulation environment is provided in a slave specific XML-file. In particular, this includes a set of capability flags to characterize the ability of the slave to support advanced master algorithms, e.g. the usage of variable communication step sizes, higher order signal extrapolation, or others.

Version 1.0.1 was released on July 10, 2017.
Version 1.0 was released on Oct. 12, 2010.

Background on FMI (2)

Functional Mockup Interface standard is going its way

- Broadly supported by simulation tools, see FMI web page: https://www.fmi-standard.org/tools
- Continuous development on future releases by a group as a Modelica Association Project
  - Members include Dassault Systems, Siemens, dSPACE, ETAS, Bosch, Daimler
- ProSTEP project to utilize FMI as standard exchange format between OEMs and suppliers
  - Members include BMW, Bosch, Conti, Daimler, Ford, MAN, several tool vendors
- Global Automotive Advisory Group for PLM support the ProSTEP initiative
  - OEM Members include BMW, Chrysler, Daimler, FIAT, Ford, GM, Nissan, Renault, VW,



ASAM

Basic idea of the joint initiative between MODELISAR and the ASAM XIL group

- Don’t develop competing standards
  - Bring together the HIL and MIL/SIL environments
  - Project proposal addresses the idea

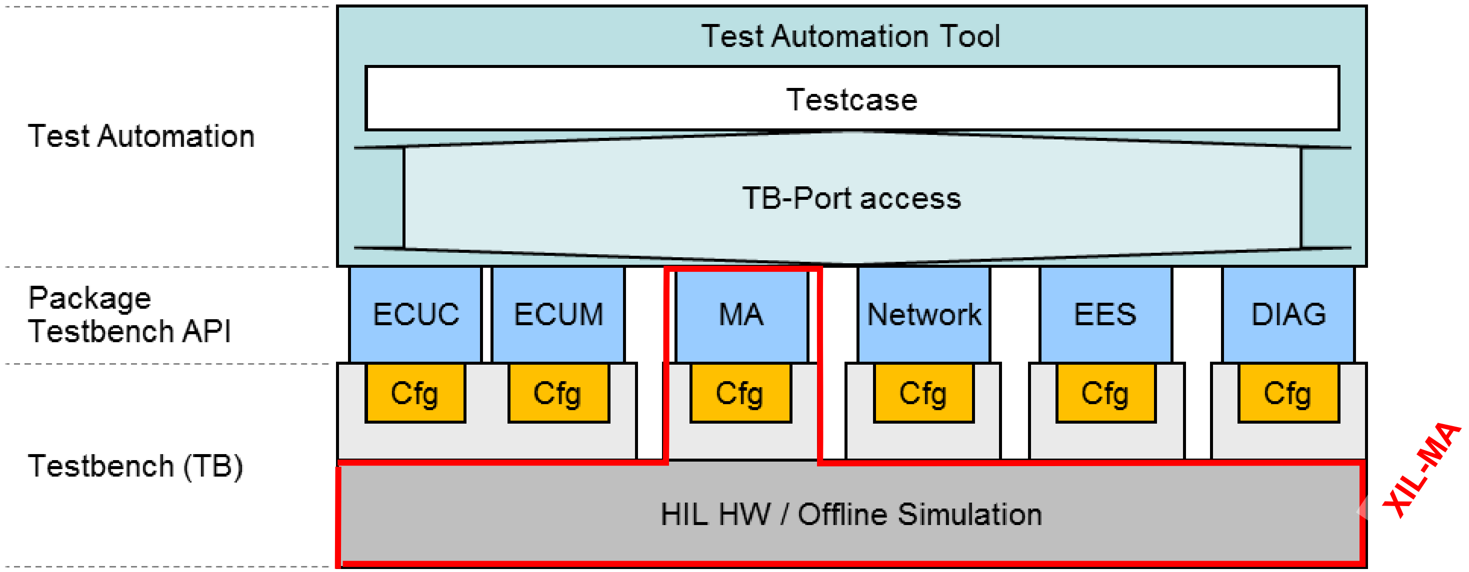
> Several companies, participating in the ITEA-2 project MODELISAR have contacted the ASAM HIL API 1.0 project team meanwhile. It had been evaluated that ASAM HIL API 1.0 functionality could and should also be used for the offline simulation scenarios of the MODELISAR use cases.
>
> Especially the simulator control API commands, which should be extended in this project will be aligned with already existing MODELISAR results.

taken from project proposal ASAM HIL 2.0.0, 2011

  - Daimler entered the ASAM HIL API project to connect the two groups
  - No effort on FMI for Application within MODELISAR
- Common understanding that results concerning offline simulation can be released as FMI for Applications
  - freely available to non-ASAM-members

General Concepts of XIL (1)

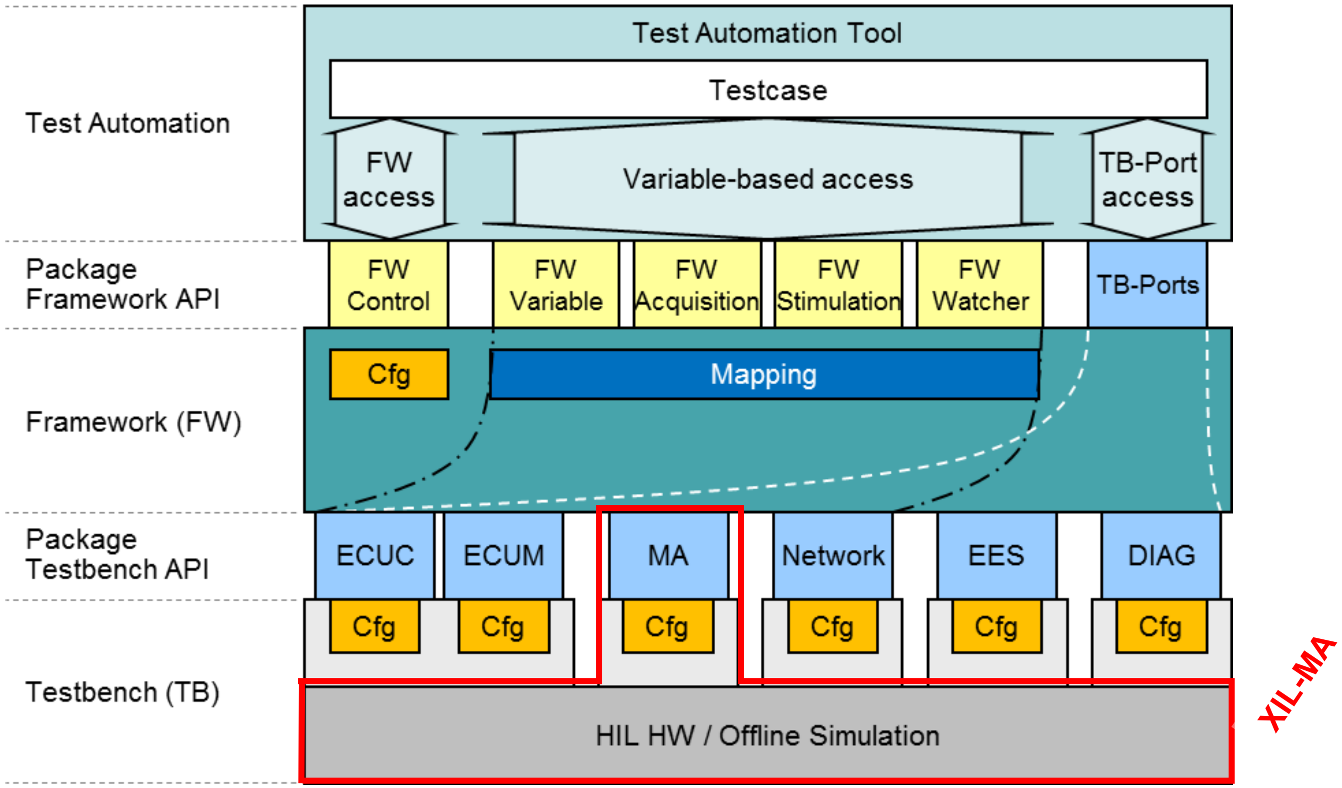
Testbench-based Access (as in HIL 1.0.2)



- XIL Removed Drawbacks of the former version HIL 1.0.2:
  - Dealing with vendor-specific start and shutdown methods (not standardized in 1.0.2)
  - Port-specific variable identifiers and data types
  - Missing overall concepts for measuring and stimulating accross different ports

General Concepts of XIL (1)

Framework-based Access (with XIL 2.0)

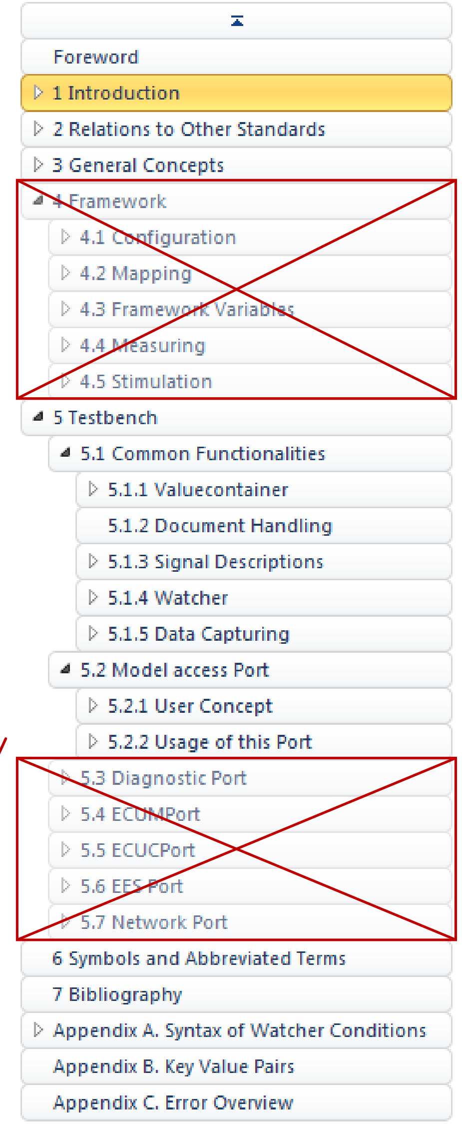


- MAPort for Simulation Model Access (XIL-MA)
  - Can be used in single testautomation applications
  - Can be integrated into the XIL Framework
    together with other ports to benefit from the full Framework functionality.

XIL-MA and XIL

Similarities and Differences

Not included in XIL-MA but in XIL



- ASAM releases an additional document:
  "Generic Simulator Interface for Simulation Model Access"
  - Contains only relevant parts **for implementation of MAPort**
  - **Freely available** for anyone as a download
  - **ASAM XIL standard remains "master"** description,
    Testbench Model Access Port and Common Functionalities
- Future releases of ASAM XIL
  - Extract the same scope of content
  - The **FMI group** should be invited to **send representatives** into XIL to contribute their experience (also non ASAM members)

Benefits of XIL-MA

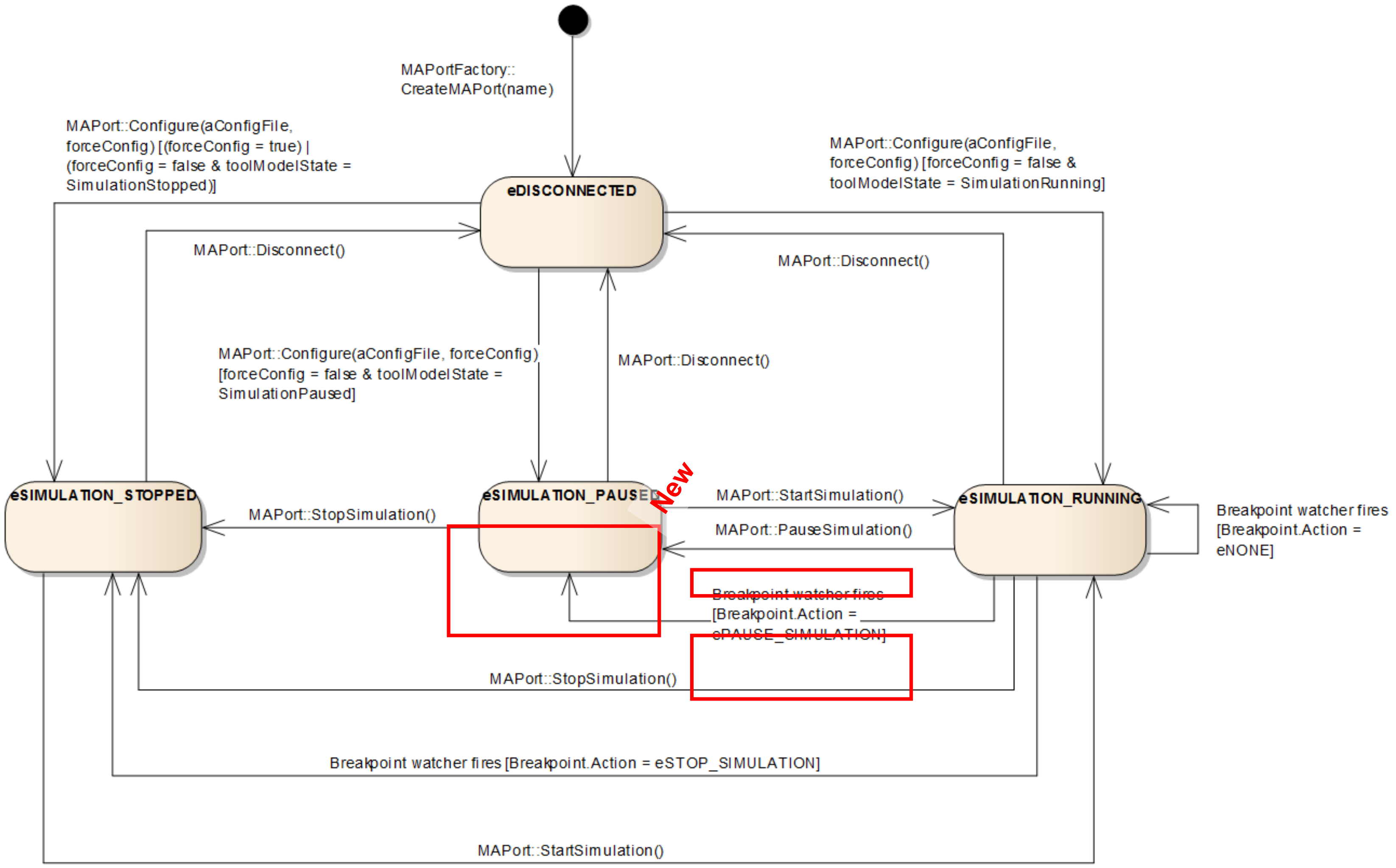
Open ASAM Document
“Generic Simulator Interface for Simulation Model Access

- Broadens scope of XIL standard to non-ASAM / non-HIL vendors
- Consistency of specification is maintained by XIL group
- ASAM ownership and copyright

What’s New?

- Pause Simulation
- Support of Real-Time Scripts
- Parameterized SignalDescriptions
- Relation between Simulation and Capturing / Stimulation
- Capture Client Events
- Read/Write Simultaneously
- Check Variable Names
- Download Parameter Sets
- Relation to FMI:
  Support of the variability tunable, fixed and const

Pause Simulation

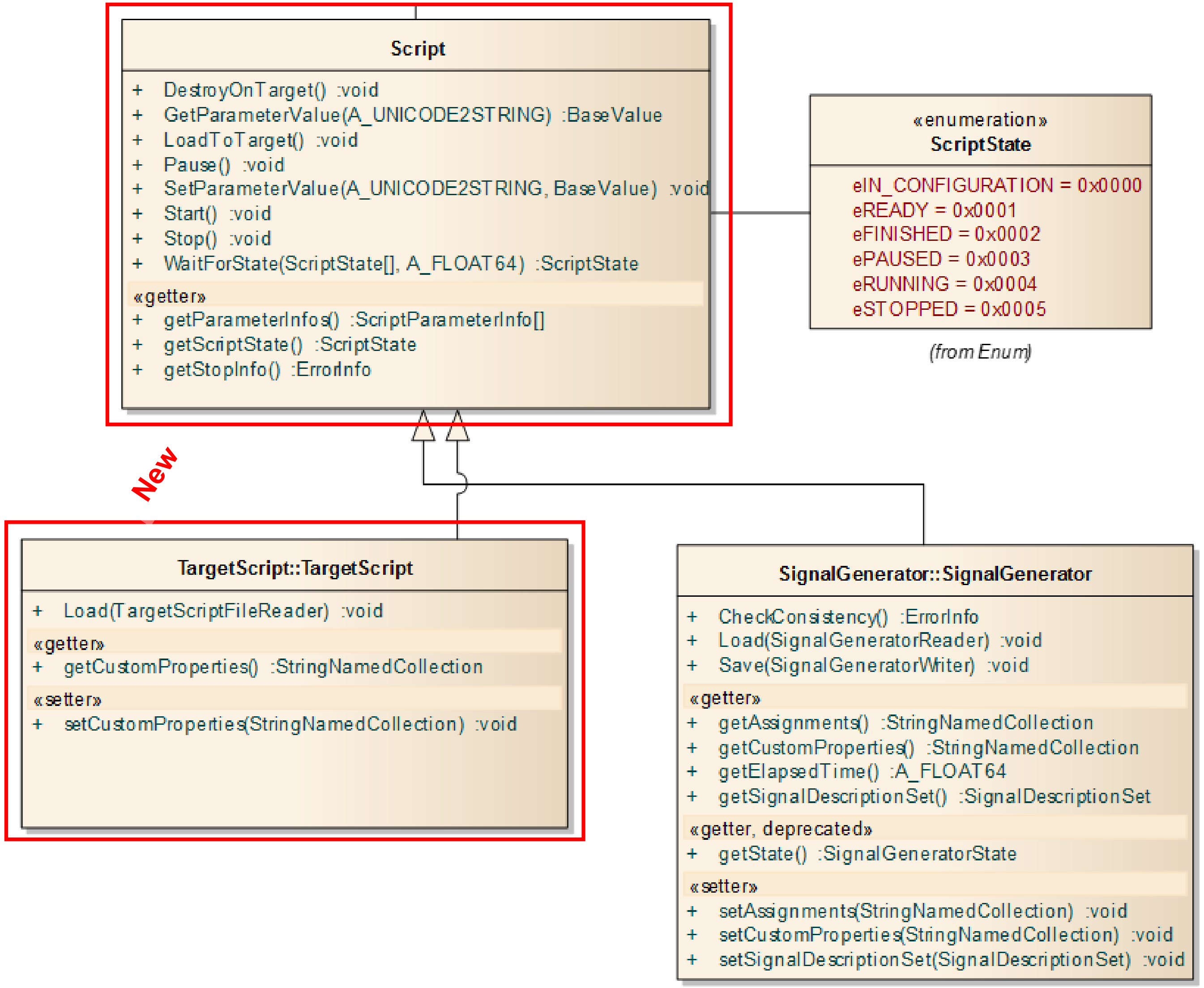


eSIMULATION_PAUSED state can be entered by either a method (PauseSimulation) or a MAPortBreakpoint

ASAM

Support of Real-Time Scripts

Users can administrate scrips (e. g. for real-time based tests) analogue to the well known SignalGenerator for stimulation, such as LoadToTarget, Start, Stop.

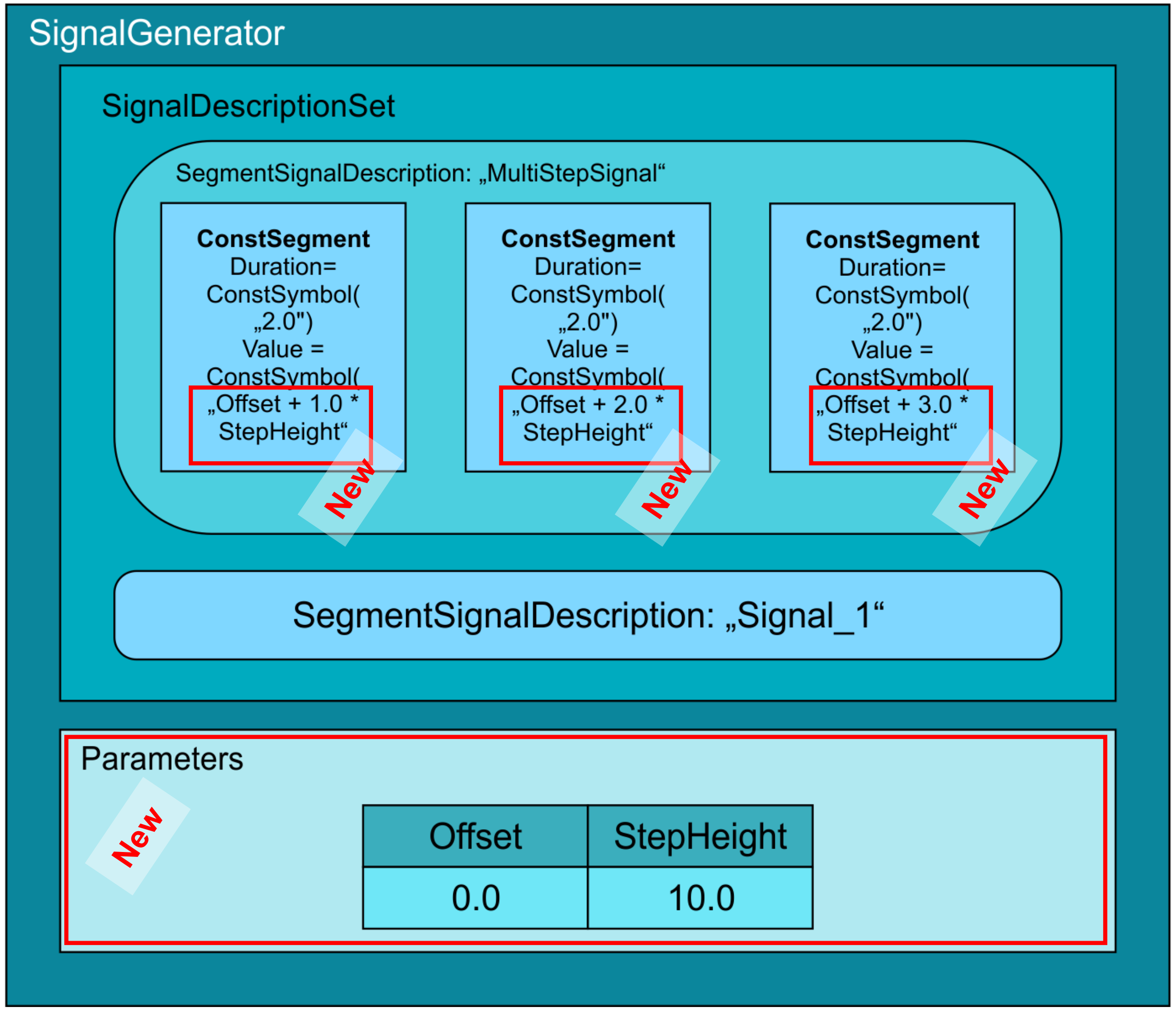


Parameterized SignalDescriptions

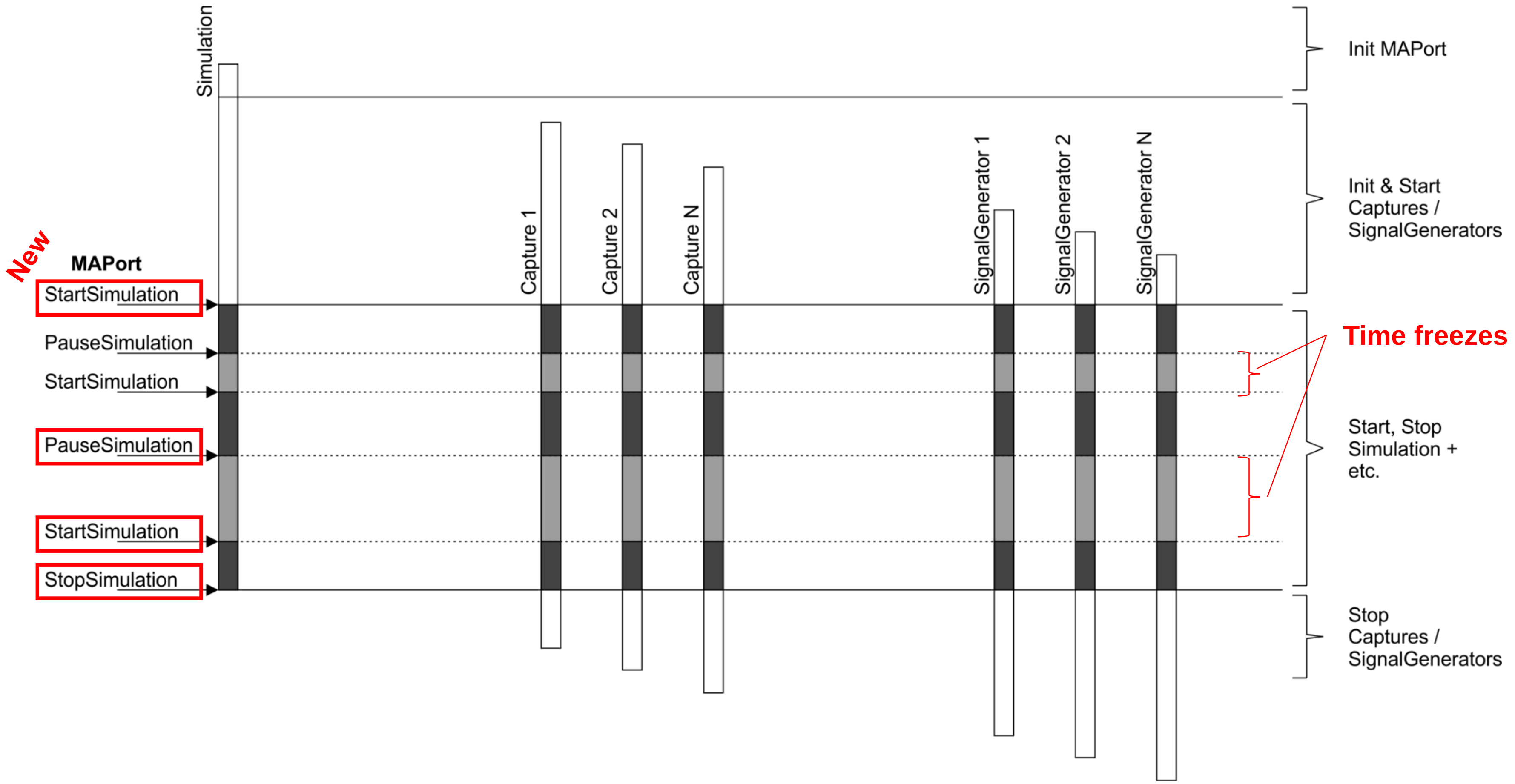
SignalDescriptions support expressions with placeholders to be assigned to SignalSegment parameters.

Benefits:
This significantly increases reusability of SignalDescriptions.

With expressions and placeholders properly applied a client does not need to know about the segment structure and the interrelations when using and adapting a predefined SignalDescription.



Relation between Simulation and Capturing / Stimulation

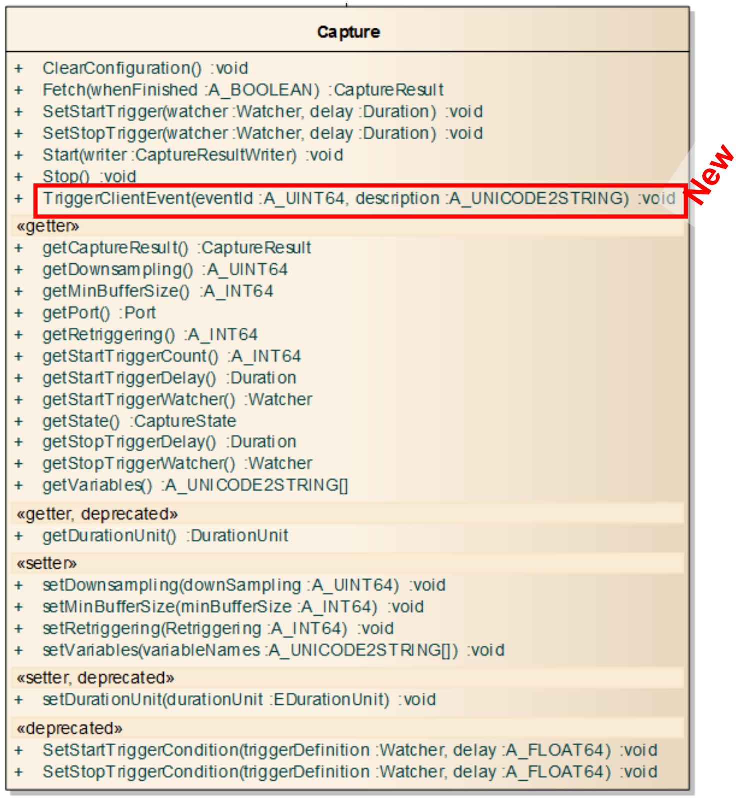


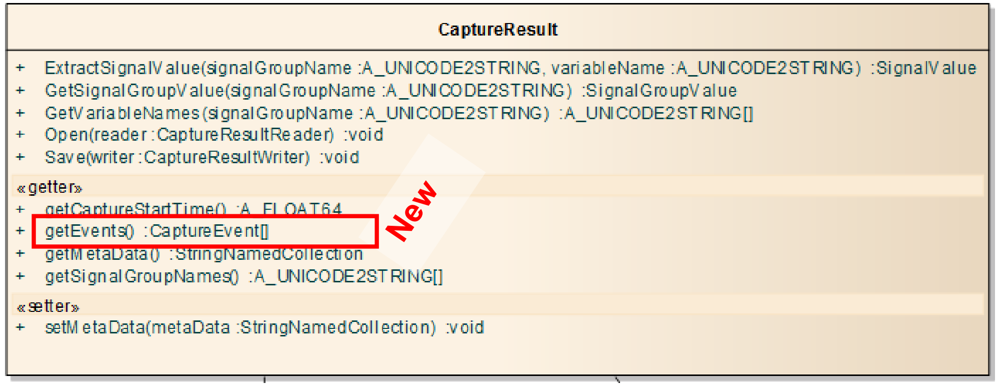
The synchronous behavior of start/stop and pause a simulation in relation to multiple captures and to stimulations has been defined precisely. For example, if a simulation pauses at the MAPort, then the simulation time freezes and so does the time in the corresponding Capture and SignalGenerator objects.

Capture Client Events

Users can add events at the Capture object, e. g. to bookmark or label interesting situations (e. g. due to electrical error simulation or feedback of the diagnostic system).

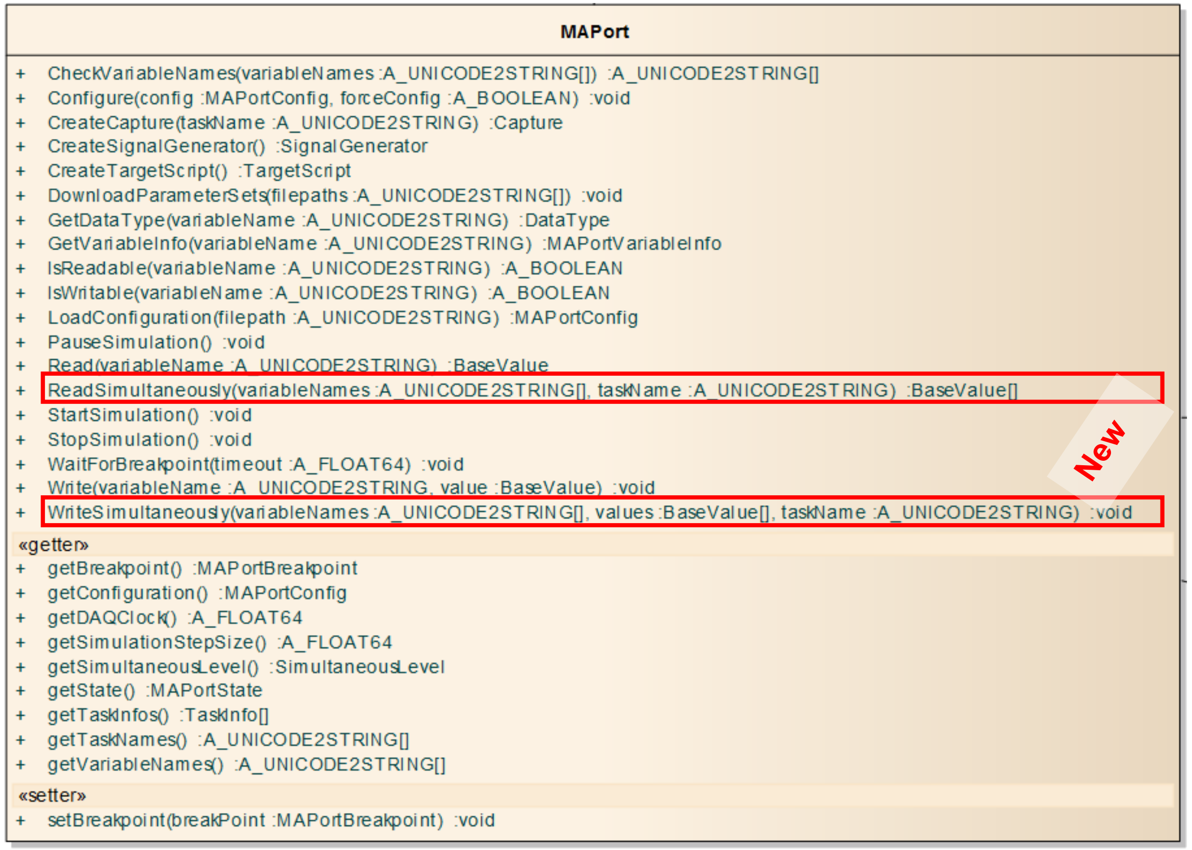
The CaptureResult object provides a getter getEvents() to access the captured events.





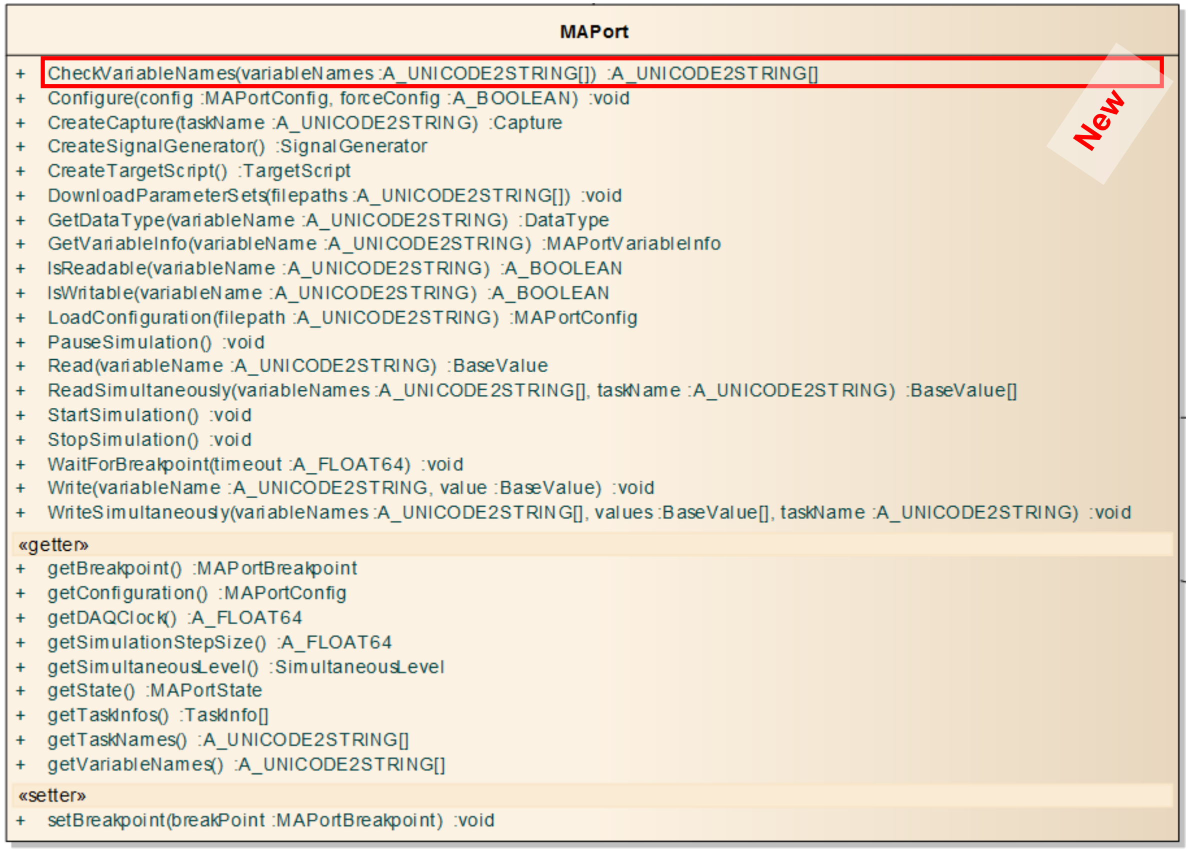
Read/Write Simultaneously

Ensure that reading and writing of multiple model variables takes place in the same simulation step or in the same cycle of a given task.



Check Variable Names

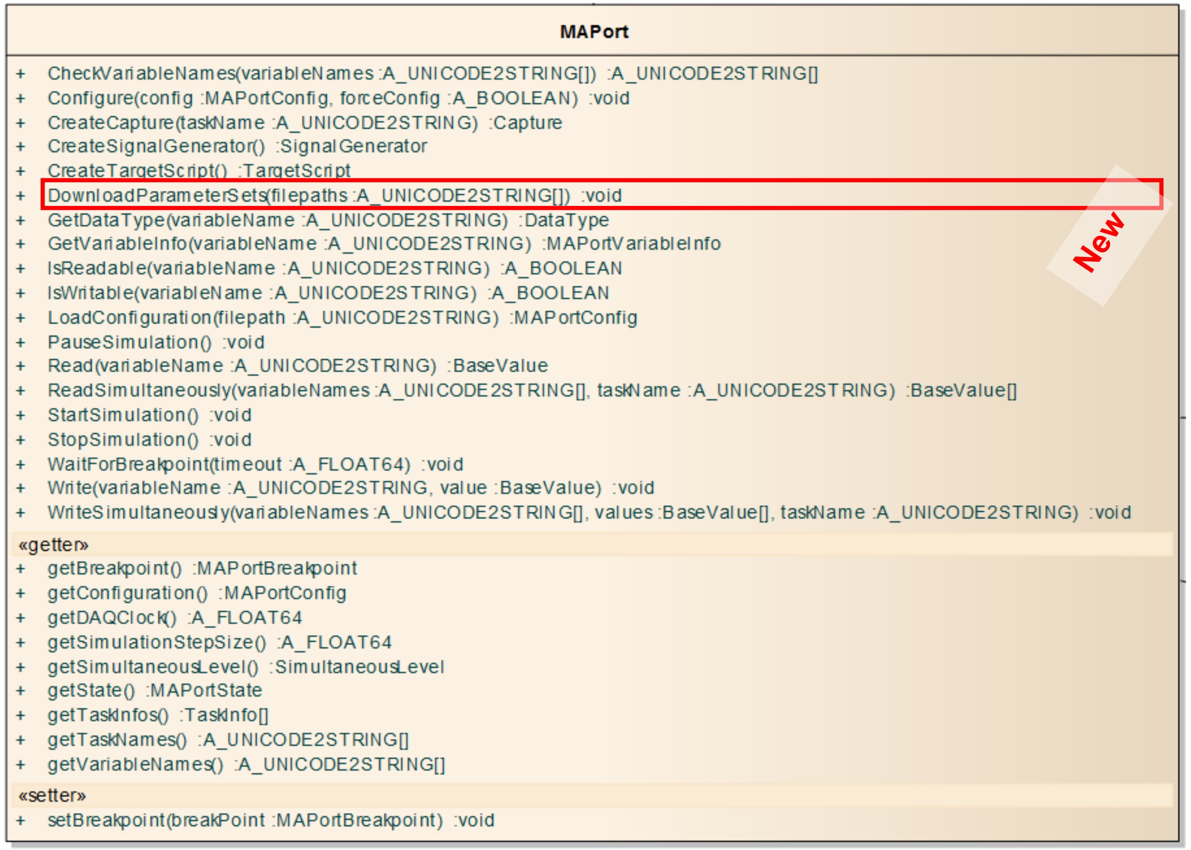
Allows the user to check, whether variable names exist or not.
The method checks if the given variable names are valid variable names with the current port configuration. Invalid names are returned. If all names are valid the returned list is empty.



ASAM

Download Parameter Sets

Loads the specified parameter set files and writes the contained parameter values to the corresponding variables of the simulation tool or hardware



Relation to FMI: Support of the variability tunable, fixed and const

The dependency of the writeability on simulation state is useful for the following reasons:

- Some variables influence a model only during initialization, e. g. initial values for discrete or continuous state variables.
- Variables can be used for computation of other variables, e. g. the vehicle's total mass m as the sum of the vehicle mass m_v and the additional load m_l. Thus, m needs to be calculated only once during initialization, and not again and again in every simulation step.

XIL already defines a MAPortVariableInfo object to check if a variable is readable or is writeable.

Existing

- MAPortVariableInfo
  - «getter» getAvailableTasks()
  - IsReadable(MAPortState)
  - IsWriteable(MAPortState)

The interpretation of the IsReadable/IsWriteable method results has been defined as described in the table according to the variability of FMI.

New

Variability according to FMI	eDISCONNECTED	eSIMULATION_STOPPED	eSIMULATION_PAUSED	eSIMULATION_RUNNING
constant	False	False	False	False
fixed	False	True	False	False
tunable	False	True	True	True

ASAM

Deliverables

Package Standard

Directory Specification

ASAM_AE_XIL-MA_AS_V2-0-1.pdf

Directory Generic UML Model

ASAM_AE_XIL-MA_AS_V2-0-1.EAP (view of test case developer)

Package Implementation Support (ASAM software parts)

Directory Templates and Directory Template Example

- Stimulus Signal Description (*.xsd, *.sti and *.stz example)
- ImplementationManifest (ImplementationManifest.xsd; xml example)

Directory Technology Reference Interfaces

Sub Directory Python contains

- Mapping_Rules (ASAM_AE_XIL_Generic-Simulatior-Interface_BS-3-4_Python-API-Technology-Reference-Mapping- Rules_V2-0-1)
- Python Interfaces (py files)

Sub Directory C# c ontains

- Mapping_Rules (ASAM_AE_XIL_Generic-Simulatior-Interface_BS-2-4_C#-API-Technology-Reference-Mapping-Rules_V2-0-1)
- Interfaces (cs files)
- Sample Code (restricted for MA-Port)

Compatibility (1)

- XIL-MA is a subset of XIL.
- The parts that are available in both standards are absolutely identical.

Compatibility (2)

The API elements listed in the second column of the table are deprecated and might be removed in a future version of the standard.

So it is recommended to use the replacement depicted in the third column.

Affected Component	Deprecated Element	Replacement
Capture (Package Testbench.Common.Capturing)	Method **SetStartTriggerCondition**	Method **SetStartTrigger**
	Method **SetStopTriggerCondition**	Method **SetStopTrigger**
	Property **DurationUnit**	Implicitly set by passing a **TimeSpanDuration** or **CycleNumberDuration** object (from package Testbench.Common.Duration) to SetStartTrigger, SetStopTrigger and the DurationWatcher factory methods.
ConstSymbol (Testbench.Common.Symbol)	Property **Value**	Property **Expression**
DurationUnit (Testbench.Common.Capturing.Enum)	Enumeration **DurationUnit**	Implicit representation by **TimeSpanDuration** and **CycleNumberDuration** in package Testbench.Common.Duration
DurationWatcher (Package Testbench.Common.WatcherHandling)	Property **Duration**	Property **Duration2**. The property's value cannot be changed. Create a new DurationWatcher for a different duration value.
SignalGenerator (Package Testbench.Common.SignalGenerator)	Property **State**	Property **ScriptState** inherited from the base interface Script in package Testbench.Common (SignalGenerator is derived from Script)
SignalGeneratorState (Package Testbench.Common.SignalGenerator.Enum)	Enumeration **SignalGeneratorState**	Enumeration **ScriptState** in package Testbench.Common.Script.Enum
SymbolFactory (Testbench.Common.Symbol)	Method **CreateConstSymbolByValue**	Method **CreateConstSymbolByExpression**
WatcherFactory (Package Testbench.Common.WatcherHandling)	Method **CreateDurationWatcher**	Methods **CreateDurationWatcherByTimeSpan** und **CreateDurationWatcherByCycleNumber**