ASAM XIL 2.2
Release Presentation

Christian Sczyrba-Neumann
dSPACE GmbH

27. February 2020
## Agenda

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Motivation for New Release</td>
</tr>
<tr>
<td>3</td>
<td>New Features</td>
</tr>
<tr>
<td>4</td>
<td>Other Changes</td>
</tr>
<tr>
<td>5</td>
<td>Backward-Compatibility</td>
</tr>
<tr>
<td>6</td>
<td>Deliverables</td>
</tr>
</tbody>
</table>
# Agenda

<table>
<thead>
<tr>
<th></th>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Motivation for New Release</td>
</tr>
<tr>
<td>3</td>
<td>New Features</td>
</tr>
<tr>
<td>4</td>
<td>Other Changes</td>
</tr>
<tr>
<td>5</td>
<td>Backward-Compatibility</td>
</tr>
<tr>
<td>6</td>
<td>Deliverables</td>
</tr>
</tbody>
</table>
Motivation Of The Standard

Separation of Test HW and Test SW by means of standardized APIs
**Concept Of Ports**

**MAPort**
Model Access port provides access to the simulation model, read and write parameters, capture and generate signals.

**NetworkPort**
Provides access to field bus systems such as CAN. E.g. Allows measurement (monitoring) and transmission (one transmit or replay) of bus data.

**EESPort**
Electrical Error Simulation port controls electrical error simulation hardware. It allows the setup of different types of errors (e.g. short cuts).

**ECUPorts (M+C)**
The ECUM port allows capturing and reading of Measurement variables. The ECUC port is used for Calibration.

**DiagPort**
Diagnostic port communicates with a diagnostic system, reads data via diagnostic services from an ECU.
Port-Based Access Of A Test Automation Tool
Variable-Based Access Of A Test Automation Tool

- Port independence of test cases by using an object-oriented access to variables
- Test Developer can use both: Testbench Port access and variable based access by the Framework
# Agenda

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Motivation for New Release</td>
</tr>
<tr>
<td>3</td>
<td>New Features</td>
</tr>
<tr>
<td>4</td>
<td>Other Changes</td>
</tr>
<tr>
<td>5</td>
<td>Backward-Compatibility</td>
</tr>
<tr>
<td>6</td>
<td>Deliverables</td>
</tr>
</tbody>
</table>
Motivation For ASAM XIL 2.2

• Address the feedback from cross tests in 2016 and 2017
• Incorporate the vendor feedback from the ASAM XIL 2.1 implementation
• Remove errors in the standard
• Close gaps in the specification about the expected behavior in certain areas
• Some new features were requested by the users (e.g. capturing support of vector and matrix data types)
## Agenda

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td>Introduction</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>Motivation for New Release</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>New Features</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>Other Changes</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td>Backward-Compatibility</td>
</tr>
<tr>
<td><strong>6</strong></td>
<td>Deliverables</td>
</tr>
</tbody>
</table>
New State Transition Of The MAPort

- Additional state transition allows to pause the simulation at the beginning
- Especially useful for MIL and SIL use-cases
Extension Of The VariableInfo Interface 1/4

ASAM XIL 2.1

The `VariableInfo` interface was not able to return the variable description and physical unit directly.

- It tooks multiple function calls to get this information in ASAM XIL 2.1.

This information will now be available directly with the new `getDescription()` and `getUnitInfo()` function.

- Improves performance and usability at the same time.

ASAM XIL 2.2

`VariableInfo` «new, getter»
+ `getCompuMethodInfo()`: CompuMethodInfo
+ `getDescription()`: A_UNICODE2STRING
+ `getUnitInfo()`: UnitInfo
+ `getDataType()`: DataType
+ `getReadable()`: A_BOOLEAN
+ `getWriteable()`: A_BOOLEAN
+ `getXSize()`: A_UINT64
+ `getYSize()`: A_UINT64

`VariableInfo` «deprecated, getter»
+ `getCompuMethodInfo()`: CompuMethodInfo
+ `getDescription()`: A_UNICODE2STRING
+ `getUnitInfo()`: UnitInfo
+ `getDataType()`: DataType
+ `getReadable()`: A_BOOLEAN
+ `getWriteable()`: A_BOOLEAN
+ `getXSize()`: A_UINT64
+ `getYSize()`: A_UINT64

`DataTypeInfo` «new»
+ `GetAxisElementType(Dimension): PrimitiveDataType`
+ `GetSize(Dimension): A_UINT64`
+ `getName()`: A_UNICODE2STRING

CompuMethod::CompuMethod «getter»
+ `getCategory()`: CompuMethodCategory

`UnitInfo` «new»
+ `GetAxisElementUnit(Dimension): A_UNICODE2STRING`
+ `getElementUnit()`: A_UNICODE2STRING

CompuMethodInfo «getter»
+ `GetAxisElementCompuMethod(Dimension): CompuMethod`
Extension Of The VariableInfo Interface 2/4

- The VariableInfo interface is now the unified location to get the variables data type and its read and writability.
- The port specific functions are not needed anymore.

**ASAM XIL 2.1**

```
VariableInfo
+ getDataType(): DataType
+ getName(): A_UNICODE2STRING
+ getReadable(): A_BOOLEAN
+ getWritable(): A_BOOLEAN
+ getXSize(): A_UINT64
+ getYSize(): A_UINT64
```

**ASAM XIL 2.2**

```
VariableInfo
+ getDescription(): A_UNICODE2STRING
```

```
DataTypeInfo
+ getContainerDataType(): ContainerDataType
+ getElementType(): PrimitiveDataType
+ getSize(Dimension): A_UINT64
+ getAxisElementType(Dimension): PrimitiveDataType
+ getElementUnit(): A_UNICODE2STRING
```

```
UnitInfo
+ getCompuMethodInfo(): CompuMethodInfo
+ getDescription(): A_UNICODE2STRING
```

**MAPort**

```
+ CheckVariableNames(variableNames: A_UNICODE2STRING[]): A_UNICODE2STRING[]
+ Configure(config: MAPortConfig, forceConfig: A_BOOLEAN): void
```

**ECUCPort**

```
+ startMeasure(): void
+ stopMeasure(): void
+ checkVariableNames(variableNames: A_UNICODE2STRING[]): A_UNICODE2STRING[]
```

**ECUMPort**

```
+ createCapture(taskName: A_UNICODE2STRING): Capture
+ configure(config: ECUMPortConfig): void
+ checkVariableNames(variableNames: A_UNICODE2STRING[]): A_UNICODE2STRING[]
```

**Deprecated**

```
+ getVariableNames(): A_UNICODE2STRING[]
+ getState(): ECUCPortState
```

```
+ getWriteable(): A_BOOLEAN
+ getREADABLE(): A_BOOLEAN
+ getWRITEABLE(): A_BOOLEAN
```

```
+ getDataType(variableName: A_UNICODE2STRING): DataType
+ getVariableNames(): A_UNICODE2STRING[]
+ getState(): ECUMPortState
```

```
+ getConfiguration(): ECUCPortConfig
+ loadConfiguration(filepath: A_UNICODE2STRING): ECUCPortConfig
```

```
+ getDAQClock(): A_FLOAT64
+ getVariableInfo(variableName: A_UNICODE2STRING): ECUCPortVariableInfo
```

```
+ createSignalGenerator(): SignalGenerator
```

```
+ configure(config: ECUCPortConfig): void
```

```
+ read(variableName: A_UNICODE2STRING): BaseValue
+ write(variableName: A_UNICODE2STRING, value: BaseValue): void
```

```
+ switchToWorkPage(): void
+ switchToRefPage(): void
```

```
+ startOnlineCalibration(loadingType: LoadingType): void
```

```
+ stopOnlineCalibration(): void
```

```
+ getVariableInfo(variableName: A_UNICODE2STRING): ECUCPortVariableInfo
```

```
+ getMeasuringVariables(): A_UNICODE2STRINGIndexCollectionNamedCollection
+ setMeasuringVariables(variables: A_UNICODE2STRINGIndexCollectionNamedCollection): void
```

```
+ loadConfiguration(filepath: A_UNICODE2STRING): ECUCPortConfig
```

```
+ getMeasuringVariables(): A_UNICODE2STRINGIndexCollectionNamedCollection
```

```
+ getConfiguration(): ECUCPortConfig
```

```
+ startMeasurement(): void
```

```
+ stopMeasurement(): void
```

```
+ getVariableNames(): A_UNICODE2STRING[]
```

```
+ getName(): A_UNICODE2STRING
```

```
+ getXSize(): A_UINT64
```

```
+ getYSize(): A_UINT64
```

```
+ getConfiguration(): ECUCPortConfig
```

```
+ loadConfiguration(filepath: A_UNICODE2STRING): ECUCPortConfig
```

```
+ getDAQClock(): A_FLOAT64
```

```
+ getVariableInfo(variableName: A_UNICODE2STRING): ECUCPortVariableInfo
```

```
+ getMeasuringVariables(): A_UNICODE2STRINGIndexCollectionNamedCollection
```

```
+ createCapture(taskName: A_UNICODE2STRING): Capture
```

```
+ configure(config: ECUCPortConfig): void
```

```
+ checkVariableNames(variableNames: A_UNICODE2STRING[]): A_UNICODE2STRING[]
```

```
+ isReadable(variableName: A_UNICODE2STRING): A_BOOLEAN
+ isWriteable(variableName: A_UNICODE2STRING): A_BOOLEAN
```
The `VariableInfo` interface was also not able to retrieve any information on the element and axis types of curve, matrix and map variables.

The new `GetDataTypeInfo()` and `GetAxisElementUnit()` function provides a way to get this information of a complex variable.
Extension Of The VariableInfo Interface 4/4

ASAM XIL 2.1

VariableInfo

- getCompuMethodInfo(): CompuMethodInfo
- getDescription(): A_UNICODE2STRING
- getUnitInfo(): UnitInfo
- getDataType(): DataType
- getReadable(): A_BOOLEAN
- getWriteable(): A_BOOLEAN
- getXSize(): A_UINT64
- getYSize(): A_UINT64
- GetDataTypeInfo(ValueRepresentation): DataTypeInfo

ASAM XIL 2.2

VariableInfo

- getCompuMethodInfo(): CompuMethodInfo
- getDescription(): A_UNICODE2STRING
- getUnitInfo(): UnitInfo
- getDataType(): DataType
- getReadable(): A_BOOLEAN
- getWriteable(): A_BOOLEAN
- getXSize(): A_UINT64
- getYSize(): A_UINT64
- GetDataTypeInfo(ValueRepresentation): DataTypeInfo

- getBySize(Dimension): A_UINT64
- getName(): A_UNICODE2STRING
- getCompuMethod(): CompuMethod
- getContainerDataType(): ContainerDataType
- getElementType(): PrimitiveDataType
- getAxisElementUnit(Dimension): A_UNICODE2STRING
- getNumericalValue(): A_INT64
- getTextValue(): A_UNICODE2STRING

- getCategory(): CompuMethodCategory

- eCustom = 0x0000
- eIdentical = 0x0001
- eLinear = 0x0002
- eRationalFunction = 0x0003
- eTexttable = 0x0004

- getDenominatorCoefficients(): A_FLOAT64[]
- getNumeratorCoefficients(): A_FLOAT64[]
- getTexttableEntries(): TexttableEntry[]

As result of this change, the user can now choose whether to read and write data from a testbench variable as raw value or physical value.

- The VariableInfo interface now distinguishes between physical value representation and raw value representation.
- It is now able to retrieve data type information on the physical value and the raw value of a testbench variable.
- It is now possible to retrieve information about the conversion method from raw value to physical value.

ASAM XIL 2.1

VariableInfo

- getCompuMethodInfo(): CompuMethodInfo
- getDescription(): A_UNICODE2STRING
- getUnitInfo(): UnitInfo
- getDataType(): DataType
- getReadable(): A_BOOLEAN
- getWriteable(): A_BOOLEAN
- getXSize(): A_UINT64
- getYSize(): A_UINT64
- GetDataTypeInfo(ValueRepresentation): DataTypeInfo

ASAM XIL 2.2

VariableInfo

- getCompuMethodInfo(): CompuMethodInfo
- getDescription(): A_UNICODE2STRING
- getUnitInfo(): UnitInfo
- getDataType(): DataType
- getReadable(): A_BOOLEAN
- getWriteable(): A_BOOLEAN
- getXSize(): A_UINT64
- getYSize(): A_UINT64
- GetDataTypeInfo(ValueRepresentation): DataTypeInfo

- getBySize(Dimension): A_UINT64
- getName(): A_UNICODE2STRING
- getCompuMethod(): CompuMethod
- getContainerDataType(): ContainerDataType
- getElementType(): PrimitiveDataType
- getAxisElementUnit(Dimension): A_UNICODE2STRING
- getNumericalValue(): A_INT64
- getTextValue(): A_UNICODE2STRING

- getCategory(): CompuMethodCategory

- eCustom = 0x0000
- eIdentical = 0x0001
- eLinear = 0x0002
- eRationalFunction = 0x0003
- eTexttable = 0x0004

- getDenominatorCoefficients(): A_FLOAT64[]
- getNumeratorCoefficients(): A_FLOAT64[]
- getTexttableEntries(): TexttableEntry[]

As result of this change, the user can now choose whether to read and write data from a testbench variable as raw value or physical value.

- The VariableInfo interface now distinguishes between physical value representation and raw value representation.
- It is now able to retrieve data type information on the physical value and the raw value of a testbench variable.
- It is now possible to retrieve information about the conversion method from raw value to physical value.
## Discard Processed Capture Data

<table>
<thead>
<tr>
<th>Capture</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ ClearConfiguration(): void</td>
</tr>
<tr>
<td>+ Fetch(whenFinished: A_BOOLEAN): CaptureResult</td>
</tr>
<tr>
<td>+ SetStartTrigger(watcher: Watcher, delay: Duration): void</td>
</tr>
<tr>
<td>+ SetStopTrigger(watcher: Watcher, delay: Duration): void</td>
</tr>
<tr>
<td>+ Start(writer: CaptureResultWriter): void</td>
</tr>
<tr>
<td>+ Stop(): void</td>
</tr>
<tr>
<td>+ TriggerClientEvent(eventId: A_UINT64, description: A_UNICODE2STRING): void</td>
</tr>
<tr>
<td>«getter»</td>
</tr>
<tr>
<td>+ getCaptureResult(): CaptureResult</td>
</tr>
<tr>
<td>+ getDiscardFetchedData(): A_BOOLEAN</td>
</tr>
<tr>
<td>+ getDownsampling(): A_UINT64</td>
</tr>
<tr>
<td>+ getMinBufferSize(): A_INT64</td>
</tr>
<tr>
<td>+ getPort(): Port</td>
</tr>
<tr>
<td>+ getRetriggering(): A_INT64</td>
</tr>
<tr>
<td>+ getStartTriggerCount(): A_INT64</td>
</tr>
<tr>
<td>«setter»</td>
</tr>
<tr>
<td>+ setDiscardFetchedData(discardFetchedData: A_BOOLEAN): void</td>
</tr>
<tr>
<td>+ setDownsampling(downSampling: A_UINT64): void</td>
</tr>
<tr>
<td>+ setMinBufferSize(minBufferSize: A_INT64): void</td>
</tr>
<tr>
<td>+ setRetriggering(retriggering: A_INT64): void</td>
</tr>
</tbody>
</table>

### Goal

Let the user decide whether to keep or discard data already retrieved by `Fetch()`

- **DiscardFetchedData == True**
  After calling `Fetch()` the data that is returned is discarded and it is not going to be returned by `getCaptureResult()` after the capturing is done.

- **DiscardFetchedData == False**
  Data returned by `Fetch()` is not discarded from memory and it will also be returned by `getCaptureResult()` after the capturing is done.

- This new option will help to avoid memory issues during long capturing sessions
Capture Support For Complex Data Types 1/2

Goal
Support the capturing of complex data types

---

`CaptureResult`

- `Open(reader: CaptureResultReader): void` (Deprecated)
- `Save(writer: CaptureResultWriter): void` (Deprecated)
- `ExtractSignalValue(signalGroupName: A_UNICODE2STRING, variableName: A_UNICODE2STRING): SignalValue`
- `GetSignalGroupValue(signalGroupName: A_UNICODE2STRING): SignalGroupValue`
- `GetVariableNames(signalGroupName: A_UNICODE2STRING): A_UNICODE2STRING[]` (getter)
- `getCaptureStartTime(): A_FLOAT64` (deprecated, getter)
- `getEvents(): CaptureEvent[]` (getter)
- `getMetaData(): StringNamedCollection` (getter)
- `getSignalGroups(): CaptureSignalGroup[]` (deprecated, getter)
- `getSignalGroupNames(): A_UNICODE2STRING[]` (getter)
- `setMetaData(metaData: StringNamedCollection): void` (getter)
Capture Support For Complex Data Types 2/2

Goal
Support the capturing of complex data types

- The new `CaptureSignalGroup` interface is now the interface to retrieve data from `CaptureResult`
- The new interface can handle scalar values as well as complex structures like `vector` and `matrix`
- The `getSignalInfos` function in the `CaptureSignalGroup()` interface enables the user also to query metadata like data type, unit, CompuMethod and more
- The `CaptureResult` can return raw values as well as physical values
Signal Generation Support For Complex Data Types 1/2

- The `SignalGenerator` interface is now able to assign a `SignalDescription` of a `SignalDescriptionSet` to more than one variable.

- **ASAM XIL 2.1**
  
  `SignalDescriptionSet1`
  - `SignalDescription_1` → `Variable_1`
  - `SignalDescription_2` → `Variable_3`

- **ASAM XIL 2.2**
  
  `SignalDescriptionSet1`
  - `SignalDescription_1` → `Variable_1`  
  - `SignalDescription_2` → `Variable_2`  
  - `SignalDescription_3` → `Variable_4`  
  - `SignalDescription_3` → `Variable_5`
Signal Generation Support For Complex Data Types 2/2

- It’s now also possible to stimulate single vector/matrix elements by signals in a `SignalGenerator`

- The `SignalGenerator` interface can now also stimulate variables with raw values or physical values

```
SignalGenerator
+ CheckConsistency(): ErrorInfo
+ Load(reader: SignalGeneratorReader): void
+ Save(writer: SignalGeneratorWriter): void

«getter»
+ getAliasDefinitions(): VariableRefNamedCollection
+ getAssignments2(): StringByVariableRefCollection
+ getCustomProperties(): StringNamedCollection
+ getElapsedTime(): A_FLOAT64
+ getSignalDescriptionSet(): SignalDescriptionSet

«getter, deprecated»
+ getAssignments(): StringNamedCollection
+ getState(): SignalGeneratorState

«setter»
+ setAliasDefinitions(aliasDefinitions: VariableRefNamedCollection): void
+ setAssignments2(assignments: StringByVariableRefCollection): void
+ setCustomProperties(properties: StringNamedCollection): void
+ setSignalDescriptionSet(value: SignalDescriptionSet): void

«setter, deprecated»
+ setAssignments(assignments: StringNamedCollection): void
```
Clarification Of Signal Descriptions / Stimulation Behavior

- Extension of the ASAM XIL 2.2 Programmers Guide to clarify:
  - Stimulation behavior at the boundary between consecutive segments of a stimulus signal
  - Stimulation behavior of idle segments
  - Stimulation behavior at the end of the stimulus signal
Get All Installed Frameworks

- It can be helpful to know all installed ASAM XIL Frameworks on a system
- The new `GetAvailableFrameworks()` function provides a list of all available ASAM XIL Frameworks
Testbench/Framework With Explicitly Specified XIL Version

• The existing implementations of
  CreateVendorSpecificTestbench()
and
  CreateVendorSpecificFramework()
will choose the implementation whose XIL version
equals the TestbenchFactory’s or FrameworkFactory’s
XIL version

• New functions provide an extra parameter that allows
the client to explicitly specify the XIL version of the
Testbench/Framework implementation to be
instantiated

• The already existing functions will continue to exist in
parallel and are not obsolete

• The new functions are only intended for special use-
cases

<table>
<thead>
<tr>
<th>TestbenchFactory</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ CreateVendorSpecificTestbench(vendorName, productName, productVersion): Testbench</td>
</tr>
<tr>
<td>+ CreateVendorSpecificTestbench2(vendorName, productName, productVersion, xilVersion): Testbench</td>
</tr>
<tr>
<td>+ GetAvailableTestbenches(): TestbenchInfo[]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FrameworkFactory</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ CreateVendorSpecificFramework(vendorName, productVersion): Framework</td>
</tr>
<tr>
<td>+ CreateVendorSpecificFramework2(vendorName, productVersion, xilVersion): Framework</td>
</tr>
<tr>
<td>+ GetAvailableFrameworks(): FrameworkInfo[]</td>
</tr>
</tbody>
</table>
Expose Framework Metadata Information

- The FrameworkMappingInfo interface didn’t provide the metadata information of the framework mapping.
- The new getMetaData() function will now provide access to this data.

---

```java
FrameworkMappingInfo

getter
+ getFcnAxisUnit(): Unit
+ getGroups(): FrameworkLabelsGroup[]
+ getLabelName(): A_UNICODE2STRING
+ getMetaData(): FrameworkMetaData
+ getTestbenchMappingInfo(): TestbenchMappingInfo
+ getType(): FrameworkVariableType
+ getXAxisUnit(): Unit
+ getYAxisUnit(): Unit

FrameworkMetaData

getter
+ getReadable(): A_BOOLEAN
+ getWriteable(): A_BOOLEAN
```
## Extension Of The Framework Configuration File 1/2

### Modification of the framework configuration file to add new parameters

- New parameters enable the framework to drive all XIL ports to all states without user interaction or default values.

### New Parameters
- ForceConfig
- ErrorConfigurationFile
- LoadingType

### Table: PortDefinition types

<table>
<thead>
<tr>
<th>PortDefinition type</th>
<th>Permitted target states</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPortDefinition</td>
<td>eDISCONNECTED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eSIMULATION_STOPPED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eSIMULATION_RUNNING</td>
<td>PortConfigurationFile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ForceConfig</td>
</tr>
<tr>
<td>DiagPortDefinition</td>
<td>eDISCONNECTED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eCONNECTED</td>
<td>PortConfigurationFile</td>
</tr>
<tr>
<td>EESPortDefinition</td>
<td>eDISCONNECTED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eCONNECTED</td>
<td>PortConfigurationFile</td>
</tr>
<tr>
<td></td>
<td>eDOWNLOADED</td>
<td>ErrorConfigurationFile</td>
</tr>
<tr>
<td></td>
<td>eACTIVATED</td>
<td></td>
</tr>
<tr>
<td>ECUCPortDefinition</td>
<td>eDISCONNECTED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eOFFLINE</td>
<td>PortConfigurationFile</td>
</tr>
<tr>
<td></td>
<td>eONLINE</td>
<td>LoadingType</td>
</tr>
<tr>
<td>ECUMPortDefinition</td>
<td>eDISCONNECTED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eMEASUREMENT_STOPPED</td>
<td>PortConfigurationFile</td>
</tr>
<tr>
<td></td>
<td>eMEASUREMENT_RUNNING</td>
<td></td>
</tr>
<tr>
<td>NetworkPortDefinition</td>
<td>eDISCONNECTED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eSTOPPED</td>
<td>PortConfigurationFile</td>
</tr>
<tr>
<td></td>
<td>eRUNNING</td>
<td></td>
</tr>
</tbody>
</table>
Extension Of The Framework Configuration File 2/2

ASAM XIL 2.1

<MAPortDefinition InstanceName="MaPortRunning" InitOrder="2" ShutdownOrder="2" TargetState="eSIMULATION_RUNNING">
    <VendorName>VendorXy</VendorName>
    <ProductName>ProductAb</ProductName>
    <ProductVersion>2019-B.2</ProductVersion>
    <PortConfigurationFile>MaPortConfig.xml</PortConfigurationFile>
</MAPortDefinition>

ASAM XIL 2.2

<MAPortDefinition InstanceName="MaPortRunning" InitOrder="2" ShutdownOrder="2">
    <VendorName>VendorXy</VendorName>
    <ProductName>ProductAb</ProductName>
    <ProductVersion>2019-B.2</ProductVersion>
    <TargetState>
        <eSIMULATION_RUNNING>
            <PortConfigurationFile>MaPortConfig.xml</PortConfigurationFile>
            <ForceConfig>false</ForceConfig>
        </eSIMULATION_RUNNING>
    </TargetState>
</MAPortDefinition>
Grouping Of Framework Labels 1/2

- ASAM XIL Framework labels can now be grouped
- Offers a way to hierarchical structure framework labels
- Groups can contain a collection of subgroups
- No artificial limitations how many groups/subgroups and framework labels are used
Grouping Of Framework Labels 2/2

## Agenda

1. Introduction
2. Motivation for New Release
3. New Features
4. **Other Changes**
5. Backward-Compatibility
6. Deliverables
Deprecation Marking In Source Code Files

**ASAM XIL 2.1**

```c
/// <summary>
/// Creates a MAPortBreakpoint object.
/// </summary>
/// <param name="watcher">Breakpoint condition (regarding elapsed simulation time
/// or certain values of simulation variables) in form of a Watcher.</param>
/// <param name="action">Action to be automatically performed when simulation
/// reaches the breakpoint, i.e when the breakpoint condition is met.</param>
int CreateMAPortBreakpoint(IWatcher watcher, BreakpointAction action);
```

**ASAM XIL 2.2**

```c
/// <summary>
/// Creates a MAPortBreakpoint object.
/// </summary>
/// <param name="watcher">Breakpoint condition (regarding elapsed simulation time
/// or certain values of simulation variables) in form of a Watcher.</param>
/// <param name="action">Action to be automatically performed when simulation
/// reaches the breakpoint, i.e when the breakpoint condition is met.</param>
[Obsolete("This method is deprecated and might be removed in a future version of
the standard. So XIL clients are strongly recommended to use the replacement
specified in the ART documentation and the Programmer's Guide Appendix.", false)]
int CreateMAPortBreakpoint(IWatcher watcher, BreakpointAction action);
```

- Deprecated functions are now also marked in the source code files as deprecated.
## Agenda

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Motivation for New Release</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>New Features</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Other Changes</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Backward-Compatibility</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Deliverables</td>
<td></td>
</tr>
</tbody>
</table>
### Backward Compatibility

<table>
<thead>
<tr>
<th>Affected Component</th>
<th>Deprecated Element</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes (Package Testbench.Common.Value Container)</td>
<td>Complete interface</td>
<td>Interface VariableInfo and its Port specific derivations (e.g. MAPortVariableInfo from package Testbench.MAPort) for predefined Attributes “Name”, “Description” and “Unit”. No replacement for user defined Attributes.</td>
</tr>
<tr>
<td>BaseValue (Package Testbench.Common.Value Container)</td>
<td>Property Attributes</td>
<td>Method GetVariableInfo of the Port Interfaces for predefined Attributes “Name”, “Description” and “Unit”. No replacement for user defined Attributes.</td>
</tr>
<tr>
<td></td>
<td>Property Type</td>
<td>Properties ContainerType and ElementName</td>
</tr>
<tr>
<td>Capture (Package Testbench.Common.Capturing)</td>
<td>Property DurationUnit</td>
<td>Implicitly set by passing a TimeSpan or CycleNumberDuration object (from package Testbench.Common.Duration) to SetStartTrigger, SetStopTrigger and the DurationWatcher factory methods.</td>
</tr>
<tr>
<td></td>
<td>Method SetStartTriggerCondition</td>
<td>Method SetStartTrigger</td>
</tr>
<tr>
<td></td>
<td>Method SetStopTriggerCondition</td>
<td>Method SetStopTrigger</td>
</tr>
<tr>
<td></td>
<td>Property Variables</td>
<td>Property Variables2</td>
</tr>
<tr>
<td></td>
<td>Method GetSignalGroupValue</td>
<td>Property SignalGroups</td>
</tr>
</tbody>
</table>

- Deprecated elements are listed in Appendix F – Deprecated Elements of the ASAM XIL 2.2 Programmers Guide
- 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.
## Agenda

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Motivation for New Release</td>
</tr>
<tr>
<td>3</td>
<td>New Features</td>
</tr>
<tr>
<td>4</td>
<td>Other Changes</td>
</tr>
<tr>
<td>5</td>
<td>Backward-Compatibility</td>
</tr>
<tr>
<td>6</td>
<td>Deliverables</td>
</tr>
</tbody>
</table>
Deliverable Content

Documents
• Content Overview
• Generic UML Model (Enterprise Architect)
• Programmers Guide

Supplementary Files
• Schemas (incl. examples)
  • EES_Configuration
  • FrameworkConfiguration
  • ImplementationManifest
  • Mapping
  • StimulusSignalDescription
• Technology Reference for C#
  • C# Interfaces
  • C# Examples
  • C# Test Suite
• Technology Reference for Python
  • Python Interfaces
• XIL Standard Assemblies (MSI Installer)

ASAM TestSuite for EESPort und MAPort