ASAM

Association for Standardisation of
Automation and Measuring Systems

ASAM AE XIL

Generic Simulator Interface

Part 2 of 4

C# API Technology Reference Mapping Rules

Version 2.1.0

Date: 2017-06-21

Base Standard

Table of Contents

FOREWORD

ASAM developed HIL API as a standard for the communication between test automation software and hardware-in-the-loop (HIL) testbenches. HIL API enables users to choose products freely according to their requirements, independent of the vendor. Several implementations of the latest version of the standard, HIL API 1.0.2, are available on the market now. This new version 2.0 of the API contains broadly extended functionality and enhanced applicability. It will support testbenches at all stages of the function software development process – MIL[1], SIL[2], HIL[3], etc. As a result, the ASAM standardization workgroup has decided to change the name to XIL API – Generic Simulator Interface with release version 2.0. After all, XIL API allows engineers to reuse their existing tests and enables a better know-how transfer from one test bench to the other, resulting in reduced training costs for employees as well.

Some areas of the XIL API standard are not HIL-specific. The MAPort, for example, can also be used to adapt simulation tools. This allows engineers to develop test cases in very early stages and in different domains in order to reuse them in later stages at a real HIL Simulator using XIL API. The Functional Mock-up Interfaces (FMI) initiative has cooperated with the XIL API Project 2.0. As a result of the ITEA2-funded project Modelisar, standardized interfaces for model exchange and cosimulation of subsystems from different domains have been developed. These “functional mock-up interfaces” will support simulation system setup at all stages of function software development (MIL, SIL, HIL, etc.).

Thus, a subset of XIL API 2.0 mainly dealing with the MAPort and simulator control as “Functional Mock-up Interface for Applications” has been released separately. This means that tests written in those early simulation environments can be directly reused in real HIL environments at a later stage.

The standard consists of different parts:

- one part for the base standard specification (programmers guide)
- one part for the mapping rules of each technology reference and
- one part for the XIL support library documentation

The base standard contains the major parts

- *Framework*, which is completely new and contains broadly extended functionality such as variable measuring and mapping as well as managing of ports already known from HIL API 1.0.2. The *Framework* chapter deals with functionality, that is based on the
- *Testbench*, which comprises the ports, known from HIL API 1.0.2. The ports were extended slightly with respect to missing functionality in the previous standard version, such as configuration and initialization. In order to give test developers standardized access to CAN busses, the Network Port was completely new designed and added to the port family.

[1] Model-in-the-Loop
[2] Software-in-the-Loop
[3] Hardware-in-the-Loop

The XIL support library contains open source software, which can be used by test case developers and framework vendors to realize vendor independent common tasks. Such tasks are

- for framework and test case
  - Unit Converter
  - Data type converter
- for Framework
  - Mapping reader (which generates a memory image from mapping file information with access possibility)
  - Creation of Framework variables (ready for usage) based on mapping information's
- for Test case
  - Realization of Framework mapping Info API
  - Mathematical operations

Factories are distributed for the generic instantiation of the framework and one or more testbenches from different vendors.

1 INTRODUCTION

1.1 OVERVIEW

ASAMs goal has always been to define technology independent standards. Therefore the object model of the XIL API is defined in UML. This UML model is mapped to different programming languages. As a result of the mapping process, all XIL API classes are available in each of the supported programming languages either as interface definitions or using native data types.

These mapping rules shall describe the transformation from the generic UML Model to C# specific interfaces, which shall be used by each client and server for implementation purpose.

1.2 DEVELOPMENT ENVIRONMENT

XIL API requires .NET Framework V4.0.
The C# project files have been created with Microsoft Visual Studio 2010.

1.3 GENERATION OF PACKAGES WITH RESPECT TO NAMESPACES

Namespaces are built according to the folder hierarchy in the UML diagram, including the package name. All namespaces start with “`ASAM.XIL`”.

1.4 DELIVERED COMPONENTS

The C# language reference includes the following components:

- XIL API interfaces
- C# example sample code

A Microsoft Visual Studio 2010 solution file is provided which contains all components.

1.4.1 XIL API INTERFACES

The C# project “ASAM.XIL.Interfaces.csproj” contains all C# interfaces required for XIL API V2.0. The file structure widely follows the namespace scheme and start with `ASAM.XIL.Interfaces.`
After compiling the project, an interface DLL ASAM.XIL.Interfaces.dll is created.
All source code is provided in C# language.

1.4.2 C# EXAMPLE SAMPLE CODE

The project "ASAM.XIL.Example.csproj” contains the source code of one or more programming examples referenced by the model specification.
The project needs references to the “ASAM.XIL.Interfaces.dll" Assembly and the Implementation Assembly of the Framework Vendor. It can serve as a starting point for end users who want to write test cases in C# using the XIL API. All source code is provided in C# language.

2 Coding and Naming Conventions

Class names, method names, attribute names, and parameter names of the C# source code have been taken directly from the UML model.

2.1 Names for Interfaces and Classes

The C# reference for XIL API strictly separates interfaces from implementation code. Therefore all classes except Exceptions of the UML diagram have been translated into C# interfaces. The C# interface names start with the capital "I", followed by the class name of the UML model. Additionally the stereotype <<interface>> is used to mark such interfaces.

Example:
UML model: `BaseValue`=> C# interface: `IBaseValue`

A corresponding C# implementation has to have the same name as the UML class name.

2.2 Properties

Methods in the UML model which start with "get" or "set" have been translated into C# properties (get, set). E.g. the methods `getMinBufferSize` and `getMaxBufferSize` of the Capture class in the UML diagram are mapped to C# as

```
long MinBufferSize { get; set; }
```

2.3 Method Signatures

2.3.1 Return Types

All return types are either of type void, C# base types or XIL-API interfaces.

2.3.2 Parameter Types

All parameter types are either C# base types or XIL API interfaces.

2.4 Enumerations

The value names of enumerations in the UML model start with "e" followed by capital characters. Enumeration classes in the generic UML model are directly mapped to C# enums. If not explicitly specified differently, numerical values start with 0.

```
    public enum DataType
    {
        eNO_TYPE = 0,
        eBOOLEAN = 1,
  ...
    }
```

3 MAPPING OF TYPES

3.1 MAPPING OF ASAM DATA TYPES

Data types of the UML class model have been mapped to the .NET type system. Basic types such as `bool` or `double` are used where possible.

Table 1 Mapping of ASAM data types to C# Data Types

UML data type	C# data type
A_BOOLEAN	bool
A_INT8	sbyte
A_INT16	short
A_INT32	int
A_INT64	long
A_UINT8	byte
A_UINT16	ushort
A_UINT32	uint
A_UINT64	ulong
A_FLOAT32	float
A_FLOAT64	double
A_UNICODE2STRING	String
A_BYTEFIELD	IList<byte>

3.2 MAPPING OF XIL TYPES

3.2.1 XIL ARRAYS

Table 2 Mapping of XIL Arrays to C# Data Types

UML data type	C# data type
<TYPE>[]	IList<TYPE>
<TYPE>[][]	IList<IList<TYPE>>

Ilist in C# has an index in the range of int.

Note: TYPE stands for signed elements with stereotype <<data type>> and standard specific interfaces (<AnyObject>).

3.2.2 XIL COLLECTIONS

Table 3 Mapping of XIL Collections to C# Data Types

UML data type	C# data type
<TYPE>NamedCollection	IDictionary<string,TYPE>
<TYPE>IndexCollection	IList<TYPE>
StringNamedCollection	IDictionary<string,string>
<TYPE>By<TYPE>Collection First Replacement Part: Value Second Replacement Part: Key	IDictionary<TYPE, TYPE> First Replacement Part: Key Second Replacement Part: Value

Note: TYPE stands for signed elements with stereotype <<data type>> and standard specific interfaces (<AnyObject>).

Note: Mapping of the <TYPE>By<TYPE>Collection changes the sequence of the replacement parts.

3.2.3 XIL ENUMERATORS

C# interfaces corresponding to iterable ASAM XIL interfaces do not inherit the `GetEnumerator` methods from the generic model. They are derived from `IEnumerable<<ANYTYPE>>` instead. Thereby `<ANYTYPE>` is replaced by the type specification of the associated `<ANYTYPE>Enumerator` interface that is given as return type of the `GetEnumerator` method. `<ANYTYPE>Enumerator` interfaces of the generic model are not mapped to any C# class.

3.2.4 XIL DATA TYPES

Table 4 Mapping of XIL Data Types to C# Data Types

UML data type	C# data type
AnyObject	dynamic
Type	Type

3.3 DISPOSABLE

Disposable will be deleted in the interface Reference. Interfaces derived from Disposable will be derived from the C# IDisposable Interface

4 EXCEPTION HANDLING

The exception handling of XIL API integrates with the standard exception handling defined by the .NET Framework. All Exception classes will be mapped to abstract Exception classes (without prefix "I" at the class name).
The class `Exception` will be deleted, the standard type `System.Exception` classes is used instead. So the implementation Classes of all Exceptions are derived from the `System.Exception`.

5 SYSTEM INITIALISATION AND CLIENT ACCESS

Entry points for client access to a XIL System are the FrameworkFactory and TestbenchFactory interface. Implementation classes of these interfaces are delivered with the ASAM XIL Technology Reference within the Implementation Package. These rely on the Implementation Manifest files provided by XIL vendors. Both factories support instance creation for different vendors at the same time.

Vendors are free to provide their own implementations of FrameworkFactory and TestbenchFactory. Please see the vendor's system documentation to figure out which factory implementation has to be used and how to get access to a vendor specific factory if provided.

The following source code listing illustrates how to obtain access to a vendor's Framework and Testbench implementation using the ASAM implementation of FrameworkFactory and TestbenchFactory.

```
using ASAM.XIL.Implementation.Framework;
using ASAM.XIL.Implementation.Testbench;

using ASAM.XIL.Interfaces.Framework;
using ASAM.XIL.Interfaces.Testbench;
using ASAM.XIL.Interfaces.Testbench.MAPort;

// creation of Framework instance
FrameworkFactory factory = FrameworkFactory();
IFramework vendorAFramework = factory.CreateFramework("vendorA", "4.4");
IFramework vendorBFramework = factory.CreateFramework("vendorB", "5.2");

// creation of Testbench and Port instances
TestbenchFactory factory = new TestbenchFactory();
ITestbench vendorATestbench = factory.CreateTestbench("vendorA", "PlatformAPI", "1.3");

IMAPort vendorAMaPort = vendorATestbench.CreateMAPort("EnvironmentModel");

ITestbench vendorBTestbench = factory.CreateVendorSpecificTestbench("vendorB", "LCO",
"5.3");
IMAPort vendorBMaPort = vendorBTestbench.CreateMAPort("EnvironmentModel")
```

6 INTERFACES AND CLASSES

Interfaces will be clearly separated from the implementation.
A client application (XIL-User) which uses the XIL API should only use the interfaces (except TestBenchFactory, FrameworkFactory). This guarantees a maximal independence from XIL API implementations coming from different tool manufacturers.
See the C# example code for more details of using the C# interfaces in own code.

7 DISTRIBUTION OF STANDARD ASSEMBLIES

7.1 INSTALLATION

To install generic ASAM assemblies (implementation and interfaces) into the Global Assembly Cache (GAC), the setup, that is delivered with the standard, has to be executed.
There is a passive mode, that allows the setup to run without interactions with user. The passive mode can be activated using the parameter “/passive”.
The ASAM setup decides, whether and which files need to be installed.

7.2 DEINSTALLATION

If a tool, that uses an implementation of the XIL API, is deinstalled, the ASAM setup must not be executed. Even if the deinstalled tool was the last consumer of the binaries, the assemblies will remain installed. The user can remove the assemblies manually using the system panel.

7.3 REPAIR

The setup has a repair mode, that can be chosen, if the setup is called once again. If installed files are missing on the computer, they are installed again.

ASAM

Association for Standardisation of
Automation and Measuring Systems

E-mail:	support@asam.net
Internet:	www.asam.net