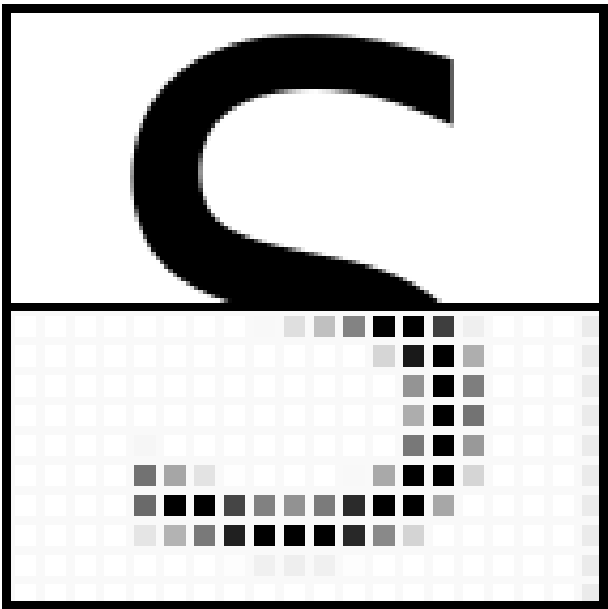


simdeck - SMP development kit - definition file



AUTHOR	Sebastien Devaux
ABSTRACT	
KEYWORDS	simulation, simulation modelling platform, ECSS-E-ST-40-07C, development kit
CONTEXT	Component simdeck-1.0.43
PROCESSED	2026-08-29 17:00:33+02:00 / sdevaux@theia

ISSUE / REVISION STATUS RECORD		
Edition	Date	Changes summary
1	2025-11-07	Document creation

Summary

1	Introduction	4
1.1	Reference	4
1.2	Scope	4
2	Requirements	4
2.1	Requirements definition	4
2.2	Some SMP Level 1 concrete classes	4
2.3	Additional utility classes	6
2.3.1	Variable length string field	6
2.3.2	Zero copy fields	6
2.3.3	Bus fields	7
3	Design	7
3.1	Zero copy fields	7
4	Validation plan	7
4.1	Strategy	7
4.2	Automated test index	7
4.2.1	Module simdeck	7
5	Developing SMP models and services with simdeck	8

1 Introduction

1.1 Reference

	Authors Reference	Title Edition
R1		<i>Simulation Modelling Platform (SMP) Level 1</i> https://ecss.nl/standard/ecss-e-st-40-07c-rev-1-simulation-modelling-platform-level-1-5-august-2025/ ECSS-E-ST-40-07C Rev.1 (5 August 2025)
R2		<i>Simulation Modelling Platform (SMP) Level 2</i> https://ecss.nl/standard/ecss-e-st-40-08c-simulation-modelling-platform-level-2-5-august-2025/ ECSS-E-ST-40-08C 5 August 2025

1.2 Scope

The simdeck component is a library of C++ classes implementing the SMP [R1] interfaces needed for models and optional services development. This is part of the simphonie project and is used for the simphonie own implementation of a SMP simulation infrastructure. Simdeck may be used with or without simphonie. It is expected to deliver with simdeck a compliant implementation of the SMP interfaces in order to let any model or service built on top of simdeck run into any SMP compliant simulation infrastructure.

This definition file includes:

- traceability requirement: identification of the implemented SMP interfaces.
- few design considerations.
- a test plan.
- a user's guide, the targeted user being a model or optional service developer that wish to deliver simulation artefact able to be hosted by a SMP simulation infrastructure.

2 Requirements

2.1 Requirements definition

The requirements in this document are defined and rendered using the following layout:

req.unique.identifier
A sentence applying the format <subject> shall <property> defining a verifiable characteristic of the subject.
Optional information providing some motivation and justification of the requirement.

Any text outside the requirement box is only additional information for a better understanding of the requirement and its scope. Such text shall not define any property to be verified. When such text is inserted between requirement, without any reference, is related to the preceding requirement.

2.2 Some SMP Level 1 concrete classes

simdeck.1
The simphonie project shall include a SMP development kit aiming at providing the concrete classes of the SMP interfaces needed to develop some components (services and models) libraries.
The SMP headers only defines pure virtual C++ classes requiring the simulator and models implementers to define every methods implementation. Many interfaces are designed to be used on both model and infrastructure side. Having the related concrete classes available speed-up the models and infrastructure development.

The SMP development kit library is expected to provide what's needed for users models and services development. It is not expected to provide any implementation for the infrastructure side components (for instance the simulator and its mandatory services implementation). Such simulation host implementation is left to the simphonie project. Even simdeck is part of the simphonie project, it may be used for any SMP component development outside simphonie.

simdeck.2

The SMP development kit shall provide a concrete class for the `Smp::IObject` interface.

simdeck.3

The SMP development kit shall provide a concrete class for the `Smp::IComponent` interface.

simdeck.4

The SMP development kit shall provide a concrete class for the `Smp::IComposite` interface.

simdeck.5

The SMP development kit shall provide a concrete class for the `Smp::ICollection` interface.

simdeck.6

The SMP development kit shall provide a concrete class for the `Smp::IContainer` interface.

simdeck.7

The SMP development kit shall provide a concrete class for the `Smp::IEntryPoint` interface.

simdeck.8.1

The SMP development kit shall provide a concrete class for the `Smp::IEntryPointPublisher` interface.

simdeck.8.2

The entry point publisher implementation shall provide a method to create an entry point directly from a void class method without argument.

Let the model or service implementer directly promote any of the component's method as an entry point that may be scheduled or subscribed to the event manager.

simdeck.9

The SMP development kit shall provide a concrete class for the `Smp::IException` interface.

simdeck.10.1

The SMP development kit shall provide a concrete classe for the `Smp::IField` interface.

simdeck.10.2

The SMP development kit shall provide a concrete classe for the `Smp::IPersist` interface.

simdeck.10.3

The SMP development kit shall provide a concrete class for the `Smp::IOutputField` interface.

simdeck.10.4

The SMP development kit shall provide a concrete class for the `Smp::ISimpleField` interface.

simdeck.10.5

The SMP development kit shall provide a concrete class implementing both `Smp::ISimpleField` and `Smp::IOutputField` interface.

simdeck.10.6

The SMP development kit shall provide a concrete class for the `Smp::IArrayField` interface.

simdeck.10.7

The SMP development kit shall provide a concrete class for the `Smp::ISimpleArrayField` interface.

simdeck.10.8

The SMP development kit shall provide a concrete class implementing both `Smp::ISimpleArrayField` and `Smp::IOutputField` interface.

simdeck.10.9

The SMP development kit shall provide a concrete class for the `Smp::IStructureField` interface.

simdeck.10.10

The SMP development kit shall provide a concrete class implementing both `Smp::IStructureField` and `Smp::IOutputField` interface.

2.3 Additional utility classes

2.3.1 Variable length string field

simdeck.addon.str.1

The simdeck library shall provide a `Smp::Publication::IType` implementation of non fixed sized string fields.

Let models and services implementers manage string parameters without the constraint of oversized fixed memory allocation.

simdeck.addon.str.2

The simdeck library shall provide a `Smp::IField` implementation to wrap a C++ `std::string` parameter.

`std::string` type is now the most common way of handling strings with C++.

simdeck.addon.str.3

The simdeck library shall provide a `Smp::IField` implementation to wrap a C++ `std::string` parameter.

`std::string` type is now the most common way of handling strings with C++.

2.3.2 Zero copy fields

simdeck.zcf.1

The simdeck library shall provide a `Smp::IOutputField` implementation able to share its data with connected fields.

The SMP publication mechanism and field handling was designed to make the model implementation quite agnostic from the simulation infrastructure, requiring the model to only share some pointers to its data. A more optimized memory management would require the model to request its memory from the simulation infrastructure to let when relevant, the infrastructure provide the same buffer pointer to be shared between a data producer and its consumers. A dedicated field type may provide the same capability.

Such buffer sharing can bring high performance increase: in many cases, the simulator spent more time to copy data from outputs to inputs than to compute anything.

The zero copy field type is compatible with itself only: it is admitted it can work only when both producer and consumer of a data are managing it using the same field class family that extends consistently the `Smp::IField` interface to do so. Such fields are necessary automatic fields, meaning the output data is pushed every time it is changed since the push finally does nothing because the same buffer is shared between the connected fields.

As well, sharing the same buffer implies the users (simulators designers) are aware such fields shall be used (read and written) sequentially only, meaning from entry points executed from the same thread.

simdeck.zcf.2

The simdeck library zero copy fields feature shall cover all the SMP primitive types.

simdeck.zcf.3

The simdeck library zero copy fields feature shall define the cast operators to let the fields type assignable to their related primitive type and the opposite.

The model implementer shall be able to use the field the same way it would use the wrapped data. It shall not require to use explicit setter or getters to access the value. The code using the fields will then remain as readable as the equivalent code using the native unboxed data.

simdeck.zcf.4

The simdeck library zero copy fields feature shall cover the simple array type for each the SMP primitive types.

2.3.3 Bus fields

The purpose of the feature is to be able to create many to one and many to many fields connections.

- define something similar to matlab/simulinks bus features.
- shall it be derive from Smp::IField (connectable by itself) or IComponent connectable with the link registry.
- quite obvious to push many outputs into a single thing. But how to select cells when connecting inputs?

3 Design

3.1 Zero copy fields

Zero copy fields are implemented the same way the "standard" fields are. The differences are:

- it manages a buffer and a pointer. The pointer defines the data wrapped by the field. When not connected, the pointer is the address of the internal buffer. When connected, the pointer is the address of the connected field's internal buffer.
- on connect, the output field push its buffer address to the target field.
- on disconnect, the target field shall be notify to restore its data pointer to its own buffer address.

Of course, the buffer pointer sharing is applied only when both source and destination are zero copy fields. Otherwise the regular data propagation process shall be applied.

4 Validation plan

4.1 Strategy

The simdeck classes shall be fully covered by unit tests. The test are implemented with the help of the cppunit library and additional helper macro from the [AcrobatomBuildSystem](https://github.com/seeduvax/AcrobatomBuildSystem)¹, to support requirement traceability management. The next section provide the unit test index.

4.2 Automated test index

4.2.1 Module simdeck

Test suite / case	Description	File / Requirements
1 - Field		/simdeck/test/TestField.cpp
1.1 - Int64		
1.2 - Int64Push		
2 - MD5		/simdeck/test/TestMD5.cpp
2.1 - Digest	Check MD5 computation using some reference digest found in wikipedia MD5 article.	
3 - SimpleArray-Field		/simdeck/test/TestSimpleArrayField.cpp
3.1 - ArrayWrapping		

¹<https://github.com/seeduvax/AcrobatomBuildSystem>

4 - SimpleVectorField		/simdeck/test/TestSimpleVectorField.cpp
4.1 - VectorWrapping		
5 - StringField		/simdeck/test/TestStringField.cpp
5.1 - SimpleTypeAssign		
6 - StructureField		/simdeck/test/TestStructureField.cpp
6.1 - CreateStructureField		
6.2 - ConnectStructFieldWithSimpleField		
6.3 - ConnectStructBetween2Field		
7 - Uuid		/simdeck/test/TestUuid.cpp
7.1 - Serialisation	Check deserialization and serialization of Uuid preserve value	
7.2 - CompareUuid		
7.3 - UuidErrorParsing	Check Error parsing and exception throwing	
8 - VectorField		/simdeck/test/TestVectorField.cpp
8.1 - VectorVectorInt		
8.2 - VectorArrayDouble		
8.3 - VectorVectorArrayDoubleOutputField		
9 - ZCSimpleField		/simdeck/test/TestZCSimpleField.cpp
9.1 - AutoPropagate		
10 - ZeroCopyField		/simdeck/test/TestZeroCopyField.cpp
10.1 - AutoPropagate		

5 Developing SMP models and services with simdeck