Final draft ETSI ES 203 119-9 V1.1.1 (2024-09)

Methods for Testing and Specification (MTS);

The Test Description Language (TDL);

Part 9: Operational Semantics and Runtime Interfaces;

<

**ETSI Standard**

Reference

DES/MTS-203119-9v111

Keywords

testing, runtime, interfaces

***ETSI***

650 Route des Lucioles

F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B

Association à but non lucratif enregistrée à la

Sous-Préfecture de Grasse (06) N° w061004871

***Important notice***

The present document can be downloaded from:
<https://www.etsi.org/standards-search>

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at [www.etsi.org/deliver](http://www.etsi.org/deliver).

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:
<https://portal.etsi.org/People/CommiteeSupportStaff.aspx>

If you find a security vulnerability in the present document, please report it through our

Coordinated Vulnerability Disclosure Program:

<https://www.etsi.org/standards/coordinated-vulnerability-disclosure>

***Notice of disclaimer & limitation of liability***

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or

other professional standard and applicable regulations.

No recommendation as to products and services or vendors is made or should be implied.

No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.

In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

***Copyright Notification***

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.
The content of the PDF version shall not be modified without the written authorization of ETSI.
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI yyyy.

All rights reserved.

Contents

Intellectual Property Rights 5

Foreword 5

Modal verbs terminology 5

Introduction 5

1 Scope 6

2 References 6

2.1 Normative references 6

2.2 Informative references 6

3 Definition of terms, symbols and abbreviations 6

3.1 Terms 6

3.2 Symbols 6

3.3 Abbreviations 6

4 Test environment 7

4.1 Architecture 7

4.2 Resource Mappings 7

4.3 Implementation Notes 7

5 Behaviour Semantics 7

5.1 Overview 7

6 Test Runtime Interfaces 7

6.1 Overview 7

6.2 Types 7

6.2.1 Overview 7

6.2.2 Argument 8

6.2.3 ComponentInstanceRole 8

6.2.4 Connection 8

6.2.5 Data 8

6.2.6 Element 9

6.2.7 ElementAnnotation 9

6.2.8 GateReference 9

6.2.9 GateTypeKind 9

6.2.10 NamedElement 10

6.2.11 Procedure 10

6.2.12 StopException 10

6.2.13 Verdict 11

6.3 System Adapter 11

6.3.1 Overview 11

6.3.2 Control Flow 11

6.3.3 Functions 12

6.4 Validator 14

6.4.1 Overview 14

6.4.2 Functions 14

6.5 Test Reporter 14

6.5.1 Overview 14

6.5.2 Functions 14

6.6 Predefined Functions 15

6.6.1 Overview 15

6.6.2 Functions 15

6.7 Helper 16

Annex A (normative or informative): Title of annex 17

Annex (informative): Bibliography 18

Annex (informative): Change history 19

History 20

# Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<https://ipr.etsi.org>).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

**DECT™**, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

# Foreword

This final draft ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS), and is now submitted for the ETSI standards Membership Approval Procedure.

The present document is part 9 of a multi-part deliverable. Full details of the entire series can be found in part 1 [1].

# Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the [ETSI Drafting Rules](https://portal.etsi.org/Services/editHelp%21/Howtostart/ETSIDraftingRules.aspx) (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

# Introduction

The present document specifies

# 1 Scope

The present document …

# 2 References

## 2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non‑specific. For specific references,only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at <https://docbox.etsi.org/Reference/>.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] [ETSI ES 203 119-1](https://www.etsi.org/deliver/etsi_es/203100_203199/20311901/): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 1: Abstract Syntax and Associated Semantics".

## 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non‑specific. For specific references,only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1]

# 3 Definition of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the [following] terms [given in ... and the following] apply:

## 3.2 Symbols

For the purposes of the present document, the [following] symbols [given in ... and the following] apply:

## 3.3 Abbreviations

For the purposes of the present document, the [following] abbreviations [given in ... and the following] apply:

# 4 Test environment

## 4.1 Architecture

## 4.2 Resource Mappings

## 4.3 Implementation Notes

TBD Provider module

# 5 Behaviour Semantics

## 5.1 Overview

# 6 Test Runtime Interfaces

## 6.1 Overview

The interfaces that should be realized and provided by end users are collectively called Test Runtime Interface (TRI).

It may provide implementations for following interfaces (listed in 'ProviderModule' class):

* 'SystemAdapter': a required component that manages interactions between runtime and SUT;
* 'Validator': a required component that provides data matching functionality;
* 'Reporter': an optional component that implements test logging;
* 'PredefinedFunctions': optional customized implementation of TDL predefined functions; and
* 'RuntimeHelper': optional customized implementation of various environment specific functions.

## 6.2 Types

### 6.2.1 Overview

The TRI consists of functional interfaces that are used to perform environment specific operations and various classes of information objects that are passed to or returned from as arguments of those operations.

The info classes are used to specify the following:

* gate and connection information for 'SystemAdapter';
* data types and values as arguments for interactions;
* test execution verdicts; and
* annotations for the above.

The classes and interfaces are defined in following clauses using Javadoc format.

TBD: type inheritance and enumeration support

TBD: ordering (currently alphabetical)

### 6.2.2 Argument

extends org.etsi.mts.tdl.execution.java.tri.Data

Extends {@link Data Data} to provide name of a parameter. The argument is specified using type and value.

Fields:

private java.lang.String **parameterName**

Name of the parameter as specified in TDL model.

Methods:

public java.lang.String **getParameterName**()

### 6.2.3 ComponentInstanceRole

extends java.lang.Enum

Predefined roles of test components as specified in TDL.

Fields:

public static final org.etsi.mts.tdl.execution.java.tri.ComponentInstanceRole

**SUT**

public static final org.etsi.mts.tdl.execution.java.tri.ComponentInstanceRole

**Tester**

Methods:

public static org.etsi.mts.tdl.execution.java.tri.ComponentInstanceRole

**valueOf**(java.lang.String name)

public static org.etsi.mts.tdl.execution.java.tri.ComponentInstanceRole[]

**values**()

### 6.2.4 Connection

extends org.etsi.mts.tdl.execution.java.tri.Element

Encapsulation of connected GateReferences.

Fields:

public final org.etsi.mts.tdl.execution.java.tri.GateReference[] **endPoints**

The end-points of this connection as specified in TDL model.

### 6.2.5 Data

extends java.lang.Object

Encapsulation of type and value. The objects are resolved using TDL data mappings.

In Java environment the type is a Java class and value is an object of that class or a lambda expression that returns such object.

Fields:

private java.lang.Object **type**

The type information in environment specific form that can be used to decode incoming data. For example, an annotated class.

private java.lang.Object **value**

The decoded value of the data that matches the type.

Methods:

public java.lang.Object **getType**()

public java.lang.Object **getValue**()

### 6.2.6 Element

extends java.lang.Object

Generic class for encapsulating information about model elements. The concrete type of the element is specified by a sub-class or the context where the element is used. Note that the element does not always represent a TDL meta-class.

Fields:

public java.util.List **annotations**

The annotations assigned to the element as specified in TDL model.

public java.lang.String **name**

The name of the element as specified in TDL model.

### 6.2.7 ElementAnnotation

extends java.lang.Object

Annotation as specified in TDL model.

Fields:

public java.lang.String **key**

Key and value of the annotation.

public java.lang.String **value**

Key and value of the annotation.

### 6.2.8 GateReference

extends java.lang.Object

Encapsulation of a connection end-point specification. It is an aggregation of following TDL model elements:

* GateInstance
* GateType
* ComponentInstance
* ComponentType
* ComponentInstanceRole

Fields:

public org.etsi.mts.tdl.execution.java.tri.Element **component**

public org.etsi.mts.tdl.execution.java.tri.Element **componentType**

public org.etsi.mts.tdl.execution.java.tri.Element **gate**

public org.etsi.mts.tdl.execution.java.tri.Element **gateType**

public org.etsi.mts.tdl.execution.java.tri.ComponentInstanceRole **role**

### 6.2.9 GateTypeKind

extends java.lang.Enum

Predefined types of gates as specified in TDL.

Fields:

public static final org.etsi.mts.tdl.execution.java.tri.GateTypeKind **Message**

public static final org.etsi.mts.tdl.execution.java.tri.GateTypeKind **Procedure**

Methods:

public static org.etsi.mts.tdl.execution.java.tri.GateTypeKind

**valueOf**(java.lang.String name)

public static org.etsi.mts.tdl.execution.java.tri.GateTypeKind[] **values**()

### 6.2.10 NamedElement

extends org.etsi.mts.tdl.execution.java.tri.Element

Extends the Element class to provide qualified name as specified in TDL.

Fields:

public java.lang.String **qualifiedName**

The qualified name of the element as specified in TDL model.

### 6.2.11 Procedure

extends org.etsi.mts.tdl.execution.java.tri.NamedElement

Encapsulation of procedure signature as specified in TDL model that may optionally be mapped to a method that implements the calling of the procedure.

Fields:

public java.util.function.Consumer **function**

Optional function that implements this procedure call.

The function parameters should match the order and types of parameters of the procedure call. That is, for each parameter of a procedure call, a method parameter should exist such that the procedure call parameter type is mapped to the method parameter type (or class).

### 6.2.12 StopException

TBD exceptions supported in all languages, maybe need a more common interaction form?

extends java.lang.Exception

The exception that is thrown when the test execution should be stopped. It is distinct from runtime exceptions as it indicates a normal completion of the test execution. If the verdict is set then it passed to {@link Validator#setVerdict(Verdict) Validator.setVerdict()} function.

This can be thrown by a {@link Validator#matches(Data, Data) Validator.matches()} when the final verdict of the test case has already been determined, for example. It is also thrown for TDL stop and timeout behaviours.

Fields:

private org.etsi.mts.tdl.execution.java.tri.Verdict **verdict**

Optional final verdict.

Methods:

public org.etsi.mts.tdl.execution.java.tri.Verdict **getVerdict**()

### 6.2.13 Verdict

extends org.etsi.mts.tdl.execution.java.tri.NamedElement

Either predefined or custom verdict.

Fields:

public static org.etsi.mts.tdl.execution.java.tri.Verdict **fail**

public static org.etsi.mts.tdl.execution.java.tri.Verdict **inconclusive**

public static org.etsi.mts.tdl.execution.java.tri.Verdict **pass**

## 6.3 System Adapter

### 6.3.1 Overview

Provides the mechanism for communicating with the system under test. This is usually a protocol stack implementation or an adapter for user interface. An implementation may choose to support multiple concurrent connections and either a message or procedure based communication (or both) depending on the testing needs. An implementation **should** be able to handle multiple receive calls (be the execution engine) for the same incoming data (see: {@link #receive(Data, Connection) receive()}. Received data should be managed using FIFO stack.

### 6.3.2 Control Flow

'SystemAdapter' implements the communication mechanisms between test execution and SUT. The interface includes methods to support both message- and procedure-based interactions. For procedure calls, the interface defines separate methods depending on whether the tester is the caller or the callee. The encoding and decoding of data is generally done by the 'SystemAdapter'.

A 'SystemAdapter' implementation is assumed to be able to handle multiple concurrent calls to 'receive' method. The implementation of the 'receive' method should block until a message is received that corresponds to the data type that was provided or the call is interrupted by the caller. This means that incoming packets should support repeated decoding attempts.

If no 'receive' calls are active when a packet arrives then the 'SystemAdapter' notifies all registered 'Receiver's and pass undecoded data to them. This also happens when none of the waiting 'receive' calls correspond to received data (that is, decoding with expected type does not succeed). The registered 'Receiver's are generally used to detect discrepancies between tester and SUT behaviour.

'ignoreUntil' is a special case of receive method, which ignores and discards (that is, does not pass to asynchronous 'Receiver's) incoming data until one arrives that matches (in terms of both type and values) the expected data. A 'Validator' instance may be used for matching.

The 'call' method blocks until a reply is received (or the call is interrupted by the caller) and it returns either the return value or an exception. It is up to the caller to determine, the semantics of the returned value. The 'receiveCall' method works similarly to the 'receive' method.

The following diagram describes an example scenario of sending a message and receiving two alternative responses and a default handling with asynchronous 'Receiver'.



Figure 9.1.2.20-1: Example of method calls involving the SystemAdapter

To avoid excessive adaptation, the execution engine supports calling procedures directly (bypassing the system adapter) if the 'ProcedureSignature's are mapped to Java methods (see clause 9.1.3). This code generation feature is configurable in settings.

### 6.3.3 Functions

Methods:

public org.etsi.mts.tdl.execution.java.tri.Data **call**(org.etsi.mts.tdl.execution.java.tri.Procedure operation, org.etsi.mts.tdl.execution.java.tri.Argument[] arguments, org.etsi.mts.tdl.execution.java.tri.Data expectedReturn, org.etsi.mts.tdl.execution.java.tri.Data expectedException, org.etsi.mts.tdl.execution.java.tri.Connection connection) throws java.lang.InterruptedException, java.lang.AssertionError

Calls a remote procedure and blocks until reply is received or exception is thrown.

Provide either expectedReturn or expectedException, but not both. When the reply/exception is received, tries to decode and match against expectedReturn or expectedException respectively.

The received reply or exception is always considered handled when this function returns.

* Parameters:
* operation - The operation to call.
* arguments - All argument values for operation parameters.
* expectedReturn - The type to be used for decoding received reply and value to match against.
* expectedException - The type to be used for decoding received exception and value to match against.
* connection - The connection on which the procedure is to be called.
* Returns:
* The reply or exception in expected format.
* Throws:
* java.lang.InterruptedException - If the waiting thread is interrupted. java.lang.AssertionError - If the received reply or exception did not match the expected data.public void **configure**(org.etsi.mts.tdl.execution.java.tri.Connection[] connections)

Prepare the adapter for all connections configured for an upcoming test description execution. This method is called before any test behaviour is executed and is a good place for doing any necessary initialization.

public org.etsi.mts.tdl.execution.java.tri.Data **receive**(org.etsi.mts.tdl.execution.java.tri.Data expected, org.etsi.mts.tdl.execution.java.tri.Connection connection) throws java.lang.InterruptedException, java.lang.AssertionError

Blocks until incoming data is available and tries to decode the first received data chunk using provided type and match against provided value.

Implementation should only consider one (the oldest) chunk of incoming data at a time. If the data matches then it is decoded and returned, otherwise the method throws **AssertionError**.

Special **null** value is used for expected parameter when the adapter should accept any data and return it (potentially without decoding). This value is only used for logging (and the format should thus be supported by the {@link Reporter Reporter} implementation).

* Parameters:
* expected - The type to be used for decoding incoming data and value to match against. If null then returns any data that is available after decoding.
* connection - The connection on which the message is to be received.
* Returns:
* The data in expected format or null if most recent data did not match the expected one.
* Throws:
* java.lang.InterruptedException - If the waiting thread is interrupted. java.lang.AssertionError - If the received data did not match the expected data.

public org.etsi.mts.tdl.execution.java.tri.Data[] **receiveCall**(org.etsi.mts.tdl.execution.java.tri.Procedure operation, org.etsi.mts.tdl.execution.java.tri.Data[] expectedArguments, org.etsi.mts.tdl.execution.java.tri.Connection connection)

public void **replyCall**(org.etsi.mts.tdl.execution.java.tri.Procedure operation, org.etsi.mts.tdl.execution.java.tri.Data reply, org.etsi.mts.tdl.execution.java.tri.Data exception, org.etsi.mts.tdl.execution.java.tri.Connection

connection)

Encode and send the reply or exception to system under test using protocol or adapter specific means.

* Parameters:
* operation - The operation to which to respond. reply - The reply to be sent.
* exception - The exception to be raised.
* connection - The connection on which the response is to be sent.

public void **send**(org.etsi.mts.tdl.execution.java.tri.Data message, org.etsi.mts.tdl.execution.java.tri.Connection connection)

Encode and send the data to system under test using protocol or adapter specific means.

* Parameters:
* message - The data to be sent.
* connection - The connection on which the message is to be sent.

## 6.4 Validator

### 6.4.1 Overview

Interface for providing custom data matching and verdict management.

### 6.4.2 Functions

Methods:

public boolean **matches**(org.etsi.mts.tdl.execution.java.tri.Data expected, org.etsi.mts.tdl.execution.java.tri.Data actual) throws java.lang.AssertionError

Performs environment specific comparison of actual data received from the system under test and the data expected by the test description.

* Returns:
* True when the objects match.
* Throws:
* java.lang.AssertionError - When actual data does not match with expected data.

public void **setVerdict**(org.etsi.mts.tdl.execution.java.tri.Verdict verdict)

Updates the current verdict of the test description execution.

## 6.5 Test Reporter

### 6.5.1 Overview

Interface for providing environment specific test logging functionality.

### 6.5.2 Functions

Methods:

public void **behaviourCompleted**(java.lang.String id)

Log the completion of a behaviour execution.

* Parameters:
* id - Locally unique identifier of the behaviour element (to be matched with {@link #behaviourStarted(String, String, Object...) behaviourStarted()}).

public void **behaviourStarted**(java.lang.String kind, java.lang.String id, java.lang.Object[] properties)

Log the start of a behaviour execution.

* Parameters:
* kind - Name of behaviour element meta-class as specified in TDL.
* id - Locally unique identifier of the behaviour element (to be matched with {@link #behaviourCompleted(String) behaviourCompleted()}).
* properties - Any behaviour specific properties.

public void **comment**(java.lang.String body)

Log comments as specified in TDL model. Comments are logged before the execution of the associated element.

public void **runtimeError**(java.lang.Throwable t)

Log a runtime error. Runtime errors result in the termination of test execution. Note that it is not the responsibility of the logger to handle the errors in any way except logging them.

* Parameters:
* t - The exception that was thrown by the execution engine or any of the adapters.

public void **testObjectiveReached**(java.lang.String uri,

java.lang.String description)

Log test objective associated with a behaviour element as specified in TDL model. Test objectives are logged after the execution of the associated behaviour.

* Parameters:
* uri - URI of the objective.
* description - Description of the objective.

## 6.6 Predefined Functions

### 6.6.1 Overview

TBD: maybe don’t list all of them here and describe general mapping for function names, parameter types etc.?

### 6.6.2 Functions

extends java.lang.Object

Default implementation of predefined functions specified in TDL.

Fields:

public org.etsi.mts.tdl.execution.java.tri.RuntimeHelper **helper**

Methods:

public boolean **and**(boolean b0, boolean b1)

public int **divide**(int i0, int i1)

public long **divide**(long i0, long i1)

public boolean **equals**(org.etsi.mts.tdl.execution.java.tri.Data d0,

org.etsi.mts.tdl.execution.java.tri.Data d1)

public boolean **gt**(int i0, int i1)

public boolean **gt**(long i0, long i1)

public boolean **gteq**(int i0, int i1)

public boolean **gteq**(long i0, long i1)

public boolean **lt**(int i0, int i1)

public boolean **lt**(long i0, long i1)

public boolean **lteq**(int i0, int i1)

public boolean **lteq**(long i0, long i1)

public int **minus**(int i0, int i1)

public long **minus**(long i0, long i1)

public int **mod**(int i0, int i1)

public long **mod**(long i0, long i1)

public int **multiply**(int i0, int i1)

public long **multiply**(long i0, long i1)

public boolean **not**(boolean b)

public boolean **notEquals**(org.etsi.mts.tdl.execution.java.tri.Data d0,

org.etsi.mts.tdl.execution.java.tri.Data d1)

public boolean **or**(boolean b0, boolean b1)

public int **plus**(int i0, int i1)

public long **plus**(long i0, long i1)

public int **size**(org.etsi.mts.tdl.execution.java.tri.Data collection)

* Parameters:
* collection - An object that a CollectionDataInstance is mapped to.

public boolean **xor**(boolean b0, boolean b1)

## 6.7 Helper

This needs a better name :)

Helper functions to provide environment specific implementation for various operations.

Methods:

public java.lang.Object **clone**(java.lang.Object object)

Create a copy of an object. Can be used to implement deep clone of {@link Data Data} objects.

public boolean **equals**(java.lang.Object o0,

java.lang.Object o1)

Compare two objects. Can be used to implement deep equals operation for {@link Data Data} objects.

* Returns:
* True if objects are equal.

Annex A (normative or informative):
Title of annex

<Text>.

Annex (informative):
Bibliography

Annex (informative):
Change history

| Date | Version | Information about changes |
| --- | --- | --- |
| <Month year> | <#> | <Changes made are listed in this cell> |
|  |  |  |
|  |  |  |

# History

|  |
| --- |
| **Document history** |
| <Version> | <Date> | <Milestone> |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

*Latest changes made on 2023-04-20*