International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.102

(06/2021)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – Specification and Description Language (SDL)

Specification and Description Language – Comprehensive SDL-2010

Recommendation ITU-T Z.102

Documents
Document Sharing Hub

Full document details: ITU-T Z.102 2021

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	**Z.100–Z.109**
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	Z.120–Z.129
User Requirements Notation (URN)	Z.150–Z.159
Testing and Test Control Notation (TTCN)	Z.160–Z.179
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-machine interfaces for the management of telecommunications networks	Z.360–Z.379
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Processing environment architectures	Z.600–Z.609

For further details, please refer to the list of ITU-T Recommendations.

Documents
Document Sharing Hub

Full document details: ITU-T Z.102 2021

Recommendation ITU-T Z.102

Specification and Description Language – Comprehensive SDL-2010

Summary

Recommendation ITU-T Z.102 defines the comprehensive features of the Specification and Description Language. Together with Recommendations ITU-T Z.100, ITU-T Z.101, ITU-T Z.103, ITU-T Z.104, ITU-T Z.105, ITU-T Z.106 and ITU-T Z.107, this Recommendation is part of a reference manual for the language. The language defined in this document covers features of the language not included in Basic SDL-2010 in Recommendation ITU-T Z.101. These features provide comprehensive coverage of abstract grammar of the language except some data features covered in ITU-T Z.104 (and ITU-T Z.107 for object-oriented data).

Coverage

The Specification and Description Language has concepts for behaviour, data description and (particularly for larger systems) structuring. The basis of behaviour description is extended finite state machines communicating by messages. Data description is based on data types for values and objects. The basis for structuring is hierarchical decomposition and type hierarchies. A distinctive feature of the Specification and Description Language is the graphical representation. This Recommendation covers additional features to Recommendation ITU-T Z.101. The concrete grammar given is the graphical representation. The alternative textual programming representation is given in Recommendation ITU-T Z.106. The concrete grammar in this Recommendation with Recommendations ITU-T Z.101 and ITU-T Z.104 (and ITU-T Z.107 for object-oriented data) gives a canonical syntax, which is extended in ITU-T Z.103 to a syntax that is easier to use. The features of the language defined in Recommendation ITU-T Z.102 make the language more comprehensive.

Applications

The Specification and Description Language is applicable within standard bodies and industry. The main application areas for which the Specification and Description Language has been designed are stated in ITU-T Z.100, but the language is generally suitable for describing reactive systems. The range of applications is from requirement description to implementation. The features of the language defined in Recommendation ITU-T Z.102 allow more complex models to be defined than is practical with ITU-T Z.101, but without the features of Recommendation ITU-T Z.103 that make the language more concise and informative. This Recommendation provides a basis for the features defined in Recommendation ITU-T Z.103, which (with ITU-T Z.104 for data and ITU-T Z.107 for object-oriented data) completes the main language feature set.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T Z.102	2011-12-22	17	11.1002/1000/11389
2.0	ITU-T Z.102	2016-04-29	17	11.1002/1000/12856
3.0	ITU-T Z.102	2019-10-14	17	11.1002/1000/14053
4.0	ITU-T Z.102	2021-06-13	17	11.1002/1000/14672

Keywords

Canonical syntax, comprehensive SDL-2010, context parameters, generic systems, inheritance, remote procedures, remote variables, SDL-2010, Specification and Description Language, specialization, state aggregation.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents/software copyrights, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the appropriate ITU-T databases available via the ITU-T website at http://www.itu.int/ITU-T/ipr/.

© ITU 2021

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

Table of Contents

		Page
1	Scope and objective	1
	1.1 Objective	1
	1.2 Application	1
2	References	1
3	Definitions	2
	3.1 Terms defined elsewhere	2
	3.2 Term defined in this Recommendation	2
4	Abbreviations and acronyms	2
5	Conventions	2
6	General rules	3
	6.1 Lexical rules – UCS extended character set	3
	6.2 End terminator and comment	3
	6.3 Empty clause	3
	6.4 Solid association symbol	3
	6.5 The metasymbol is followed by and flow line symbols	3
	6.6 Visibility rules, names and identifiers – additional scope units	3
	6.7 Macro	4
7	Organization of Specification and Description Language specifications	6
	7.1 Framework	6
	7.2 Package	6
8	Structural concepts	7
	8.1 Types, instances and gates	7
	8.2 Type references and operation references	12
	8.3 Context parameters	13
	8.4 Specialization	19
9	Agents	23
	9.1 System	24
	9.2 Block	25
	9.3 Process	25
	9.4 Procedure	25
10	Communication	27
	10.1 Channel	27
	10.2 Connection	27
	10.3 Signal	27
	10.4 Signal list area	28
	10.5 Remote procedure	28
	10.6 Remote variable	32

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

Page

11 Behaviour 36
11.1 Start 36
11.2 State 36
11.3 Input 38
11.4 Priority Input 39
11.5 Continuous signal 40
11.6 Enabling condition 41
11.7 Save 41
11.8 Empty clause 41
11.9 Spontaneous transition 41
11.10 Label (connector name) 42
11.11 State machine and composite state 42
11.12 Transition 49
11.13 Action 51
11.14 Statement lists 53
11.15 Timer 62
12 Data 62
12.1 Data definitions 62
12.2 Use of data 63
12.3 Active use of data 63
13 Generic system definition 64
13.1 Optional definition 64
13.2 Optional transition string 66

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

Introduction

Status/Stability

This Recommendation is part of the ITU-T Z.100 to ITU-T Z.107 series of Recommendations that give the complete language reference manual for SDL-2010. The text of this Recommendation is stable. For more details see Recommendation ITU-T Z.100.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

Recommendation ITU-T Z.102

Specification and Description Language – Comprehensive SDL-2010

1 Scope and objective

This Recommendation defines features of the Specification and Description Language that extend the features of the Basic Specification and Description Language defined in [ITU-T Z.101] to comprehensively cover the abstract grammar of the language. A canonical concrete syntax is given to cover the abstract grammar. The abstract grammar defined in this document with the abstract grammar of the Basic Specification and Description Language comprehensively defines properties of the language except the details of data and expressions. This Recommendation also includes features for inheritance of types, generic systems, macros and the handling of the Universal Multiple-Octet Coded Character Set. Together with [ITU-T Z.100], [ITU-T Z.101], [ITU-T Z.103], [ITU-T Z.104], [ITU-T Z.105], [ITU-T Z.106] and [ITU-T Z.107], this Recommendation forms a reference manual for the language.

1.1 Objective

The objective of this Recommendation is to define features of the Specification and Description Language additional to those features defined for Basic SDL-2010 so that the abstract grammar of all SDL-2010 except data and expressions is defined. As well as completing the abstract grammar, the features for the inheritance of types is added. The defined concrete syntax is a canonical form that closely matches the abstract grammar. While this makes it possible to use each of the features, a separate Recommendation on shorthand notation provides additional concrete grammar that makes SDL-2010 more practical to use.

1.2 Application

This Recommendation is part of the reference manual for the Specification and Description Language. The part of the language defined by this Recommendation does not usually include shorthand notation or *Model* sections, so that a model written using only Comprehensive SDL-2010 or Basic SDL-2010 is not as concise or as readable as one using the full language. Shorthand notation or *Model* sections in Comprehensive SDL-2010 are transformed into the canonical concrete syntax given for Comprehensive SDL-2010 or Basic SDL-2010 or the data and action language in SDL-2010.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T T.50] Recommendation ITU-T T.50 (1992), *International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) – Information technology – 7-bit coded character set for information interchange*.

[ITU-T Z.100] Recommendation ITU-T Z.100 (2021), *Specification and Description Language – Overview of SDL-2010*.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

[ITU-T Z.101] Recommendation ITU-T Z.101 (2021), *Specification and Description Language – Basic SDL-2010*.

[ITU-T Z.103] Recommendation ITU-T Z.103 (2021), *Specification and Description Language – Shorthand notation and annotation in SDL-2010*.

[ITU-T Z.104] Recommendation ITU-T Z.104 (2021), *Specification and Description Language – Data and action language in SDL-2010*.

[ITU-T Z.105] Recommendation ITU-T Z.105 (2021), *Specification and Description Language – SDL-2010 combined with ASN.1 modules*.

[ITU-T Z.106] Recommendation ITU-T Z.106 (2021), *Specification and Description Language – Common interchange format for SDL-2010*.

[ITU-T Z.107] Recommendation ITU-T Z.107 (2021), *Specification and Description Language – Object-oriented data in SDL-2010*.

[ITU-T Z.111] Recommendation ITU-T Z.111 (2016), *Notations and guidelines for the definition of ITU-T languages*.

[ISO/IEC 10646] ISO/IEC 10646:2020, *Information technology – Universal Coded Character Set (UCS)*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

The definitions of [ITU-T Z.100] apply.

3.2 Term defined in this Recommendation

None.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

The abbreviations and acronyms defined in [ITU-T Z.100] apply.

5 Conventions

The conventions defined in [ITU-T Z.100] apply, which includes the conventions defined in [ITU-T Z.111].

Where an abstract or concrete syntax rule is defined in this Recommendation with the same name as a rule in [ITU-T Z.101] the rule given here replaces the rule in [ITU-T Z.101]. Any *Abstract grammar* or *Concrete grammar* conditions, *Semantics* and *Model* defined on a named rule in [ITU-T Z.101] apply to the redefined rule, unless specifically defined otherwise in this Recommendation. Any contradiction between [ITU-T Z.101] and this Recommendation is an error in the definition of SDL-2010 that needs to be resolved by further study.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

6 General rules

6.1 Lexical rules – UCS extended character set

The use of the extended character set of UCS (see [ISO/IEC 10646]) is allowed. UCS includes the International Reference Version (IRV) characters (see [ITU-T T.50] and [ITU-T Z.101]) as a subset.

If the UCS extended character set is used, the printing characters that are not defined by IRV are permitted to appear freely in a <character string> in a <comment> or within a <note>. A printing character of the UCS extended character set that corresponds to an IRV <letter> is equivalent to the IRV <letter>. Similarly, a printing character of the UCS extended character set that corresponds to an IRV <decimal digit>, <special>, <other special> or <other character> is equivalent to the IRV <decimal digit>, <special>, <other special> or <other character>, respectively. A printing character of the UCS extended character set that represents a letter in some script and does not correspond to an IRV <letter> is usable as a <letter>. Characters are treated in the order they occur in the source text, which possibly does not correspond to the apparent presentation or printing order depending on how UCS characters are handled for presentation or printing (such as right to left, left to right, or in combination). Non-printing UCS characters that do not correspond to an IRV character are treated in the same way as a control character is treated in [ITU-T Z.101].

6.2 End terminator and comment

Concrete grammar

An <end> in <statements> shall not contain <comment>.

6.3 Empty clause

This clause is intentionally left blank.

6.4 Solid association symbol

See [ITU-T Z.101].

6.5 The metasymbol is followed by and flow line symbols

See [ITU-T Z.101].

6.6 Visibility rules, names and identifiers – additional scope units

This Recommendation introduces concrete syntax for <compound statement>, which is a scope unit for the connector name and variables introduced in the statement.

A <signal definition> is a scope unit only for any formal context parameters of the defined signal. These context parameters are formal sorts that are replaced by actual sorts when the signal is specialized. Such specialization is not part of Basic SDL-2010 but is described in this Recommendation.

NOTE – A <sort context parameter> is not considered to be a scope unit in SDL-2010. It was listed as defining a scope unit in SDL-2000.

A formal context parameter is an entity of the same entity kind as the corresponding actual context parameters.

Abstract grammar

Path-item	=	*Package-qualifier*
	|	*Agent-type-qualifier*
	|	*Agent-qualifier*
	|	*State-type-qualifier*
	|	*State-qualifier*

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

| | *Data-type-qualifier*
| | *Procedure-qualifier*
| | *Interface-qualifier*
| | *Compound-node-qualifier*

Compound-node-qualifier :: *Interface-name*

Compound-node-name = *Name*

The abstract syntax is extended from Basic SDL-2010 to include *Compound-node-qualifier* in *Path-item* to identify the scope of a compound statement.

Concrete grammar

<scope unit kind> ::=

package
| **system type**
| **system**
| **block**
| **block type**
| **process**
| **process type**
| **state**
| **state type**
| **procedure**
| **signal**
| **type**
| **operator**
| **method**
| **interface**
| **composition**

The syntax of <scope unit kind> is extended from Basic SDL-2010 to include **composition** in Path-item to identify the scope of a compound statement. If the <scope unit kind> of a <qualifier> is **composition**, the <qualifier> a represents a *Compound-node-qualifier* and <name> of the <qualifier> is the <connector name> of a <compound statement>. If no <connector name> is given for the <compound statement>, a newly created anonymous name represents the *Connector-name*; therefore, no explicit <qualifier> can be given. In this case the specification is only valid if all uses of a variable name defined in the compound statement are uniquely bound without a qualifier.

6.7 Macro

A macro definition contains a collection of lexical units, which are included as macro expansions in the places in the textual parts of the concrete grammar of an <sdl specification>. Each such place is indicated by a macro call. Before an <sdl specification> can be analysed, each macro call shall be replaced by the corresponding macro expansion.

6.7.1 Additional lexical rules

<formal name> ::=

[<name or number>**%**] <macro parameter>
{ [**%**<name or number>] **%**<macro parameter> }*
[**%**<name or number>]

6.7.2 Macro definition

<macro definition> ::=

macrodefinition <macro name>
[<macro formal parameters>] <end>
<macro body>
endmacro [<macro name>] <end>

<macro formal parameters> ::=

(<macro formal parameter> { **,** <macro formal parameter>}* **)**
| **fpar** <macro formal parameter> {, <macro formal parameter>}*

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

NOTE – The two alternatives of <macro formal parameters> are equivalent.

<macro formal parameter> ::=
<name>

<macro body> ::=
{<lexical unit> | <formal name>}*

<macro parameter> ::=
<macro formal parameter>
| **macroid**

Each <macro formal parameter> of a <macro definition> shall have a distinct <name> within the macro. Each <macro actual parameter> of a macro call shall be matched one to one by position with the corresponding <macro formal parameter>.

The <macro body> shall not contain the keyword **endmacro** or **macrodefinition**.

A <macro definition> contains lexical units.

A <macro name> is visible in the whole system definition, no matter where the macro definition appears. A macro call is allowed to appear before the corresponding macro definition.

A macro definition is allowed to contain macro calls, but a macro definition shall not call itself either directly or indirectly through macro calls in other macro definitions.

The keyword **macroid** is allowed as a pseudo macro formal parameter within each macro definition. No <macro actual parameter> item is allowed for the keyword **macroid** as a pseudo macro formal parameter, and the keyword is replaced by a unique <name> for each expansion of a macro definition (within an expansion, the same <name> is used for each occurrence of **macroid**).

Places that a <macro definition> is allowed are shown in *Concrete grammar*. A <macro definition> is not removed once macro expansion has taken place and therefore acts as a separator between other lexical units.

Example

Below is an example of a <macro definition>:

```
macrodefinition Exam (alfa, c, s, a);
     block type alfa referenced;
     dcl exported c as s Integer := a;
endmacro Exam;
```

6.7.3 Macro call

<macro call> ::=
macro <macro name> [<macro call body>] <end>

<macro call body> ::=
(<macro actual parameter> {**,** <macro actual parameter>}* **)**

<macro actual parameter> ::=
<lexical unit>*

The <lexical unit> for a <macro actual parameter> is not allowed to be a comma "," or right parenthesis ")". If either of these two characters is required in a <macro actual parameter>, the <macro actual parameter> has to be a <character string>. If the <macro actual parameter> is a <character string>, the result of the <character string> is used when the <macro actual parameter> replaces a <macro formal parameter>.

A <macro call> is allowed to appear at any place where a <lexical unit> is allowed.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

Because a <macro call> is replaced by other lexical units once macro expansion has taken place, it does not occur in *Concrete grammar*.

Model

An <sdl specification> is allowed to contain macro definitions and macro calls. Before such an <sdl specification> is analysed for the remaining non-macro grammar, all macro calls shall be expanded. The expansion of a macro call means that a copy of the macro definition having the same <macro name> as that given in the macro call is expanded to replace the macro call. This means that a copy of the macro body is created, and each occurrence of the <macro formal parameter>s of the copy is replaced by the corresponding <macro actual parameter>s of the macro call, then macro calls in the copy, if any, are expanded. All percent characters (%) in <formal name>s are removed when <macro formal parameter>s are replaced by <macro actual parameter>s.

There shall be a one to one correspondence between <macro formal parameter> and <macro actual parameter>.

Example

Below is an example of a <macro call>, as a fragment inside a <text symbol> of a <block type diagram>.

```
.........
block type A referenced;
macro Exam (B, C1, S1, 12);
.........
```

The expansion of this macro call, using the example in clause 6.7.2, gives the following result.

```
.........
block type A referenced;
block type B referenced;
dcl exported C1 as S1 Integer := 12;
.........
```

7 Organization of Specification and Description Language specifications

7.1 Framework

See [ITU-T Z.101].

7.2 Package

Definitions as parts of a package are extended to include remote items and generic selection (in addition to defining types, signals, and interfaces as defined in Basic SDL-2010).

Concrete grammar

<package text area> ::=
<text symbol> ***contains***
{ <signal definition list>
| <data definition>
| <remote procedure definition>
| <remote variable definition>
| <macro definition>
| <select definition>}*

<package text area> is extended to cover <remote variable definition>, <remote procedure definition>, <select definition> and <macro definition>. Each <remote procedure definition> or <remote variable definition> in a <package text area> represents a member of the *Signal-definition-**set*** of the *Package-definition* for the implied signal of the remote procedure or variable. The content of a <macro definition> is used for macro calls, and the macro expansion represents the abstract grammar items in the context the macro call occurs. Only the selected items of a <select definition> represent items on the abstract grammar.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

<diagram in package> ::=
| <package reference area>
| <entity in agent diagram>
| <option area>

<diagram in package> is extended from Basic SDL-2010 to cover <option area> for generic specifications. Only the selected items of an <option area> represent items on the abstract grammar.

<selected entity kind> ::=
system type
| **block type**
| **process type**
| **package**
| **signal**
| **procedure**
| **type**
| **state type**
| **synonym**
| **signallist**
| **interface**
| **remote procedure**
| **remote**

The <selected entity kind> used in <definition selection> of a <package use clause>, is extended from Basic SDL-2010 to include **remote procedure** and **remote** for remote procedures and remote variable definitions.

The <selected entity kind> **procedure** is used for selection of both (normal) procedures and remote procedures. If both a normal procedure and a remote procedure have the given <name>, **procedure** denotes the normal procedure. To force the <definition selection> to denote the remote procedure, the **procedure** keyword is optionally preceded by **remote**.

The keyword **remote** (not preceding **procedure**) is used for selection of a remote variable definition.

8 Structural concepts

Basic SDL-2010 is extended to include the generalization and specialization of types.

The language mechanisms introduced provide:

a) (pure) type definitions as defined in Basic SDL-2010;

b) typebased instance definitions that define instances or instance sets according to types as in Basic SDL-2010;

c) parameterized type definitions that are made (partially or completely) independent of the enclosing scope by means of context parameters and are bound to specific scopes by actual parameters;

d) specialization of supertype definitions into subtype definitions, by adding properties and by redefining virtual types and transitions.

8.1 Types, instances and gates

The type of an instance (or instance set) based on a parameterized type is the anonymous type formed by binding the parameters of the parameterized type to the actual parameters given.

8.1.1 Structural type definitions

Basic SDL-2010 is extended by the introduction of virtuality, context parameters and the state aggregation type.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

8.1.1.1 Agent types

Abstract grammar

Agent-type-definition :: *Agent-type-name*
Agent-kind
[*Agent-type-identifier*]
*Agent-formal-parameter**
*Data-type-definition-**set***
*Syntype-definition-**set***
*Signal-definition-**set***
*Timer-definition-**set***
*Variable-definition-**set***
*Agent-type-definition-**set***
*Composite-state-type-definition-**set***
*Procedure-definition-**set***
*Agent-definition-**set***
*Gate-definition-**set***
*Channel-definition-**set***
State-machine
[*Abstract*]

Agent-type-definition is extended to optionally allow *Abstract* to be specified.

Concrete grammar

<type preamble> ::=
[<virtuality> [<abstract>] | <abstract> [<virtuality>]]

<type preamble> is extended from the empty syntax in Basic SDL-2010 to optionally allow <virtuality> and <abstract> to be specified.

<agent type additional heading> ::=
[<formal context parameters>] [<virtuality constraint>]
<agent additional heading>

<agent type additional heading> is extended from Basic SDL-2010 to allow <formal context parameters> and <virtuality constraint> to be specified. If <agent type additional heading> has <formal context parameters>, the agent type is an abstract type: that is, *Abstract* is included in the *Agent-type-definition* (regardless of the keyword **abstract** in the <type preamble>).

<agent additional heading> ::=
[<specialization>] [<agent formal parameters>]

<agent additional heading> is extended compared with Basic SDL-2010 to allow <specialization> for formal context parameters.

Semantics

The complete output set of an agent type includes in the union of all signals, the implicit signals for remote procedures and remote variables mentioned, either directly or as part of interfaces, in the outgoing signal lists associated with the gates of the agent type.

8.1.1.2 System type

Concrete grammar

A <formal context parameter> of <formal context parameters> of <agent type additional heading> of a <system type diagram> shall not be an <agent context parameter>, a <variable context parameter list> or a <timer context parameter list>.

8.1.1.3 Block type

See [ITU-T Z.101].

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

8.1.1.4 Process type

See [ITU-T Z.101].

8.1.1.5 Composite state type

Abstract grammar

Composite-state-type-definition :: *State-type-name*
[*Composite-state-type-identifier*]
*Composite-state-formal-parameter****
*State-entry-point-definition-**set***
*State-exit-point-definition-**set***
*Gate-definition-**set***
*Data-type-definition-**set***
*Syntype-definition-**set***
*Composite-state-type-definition-**set***
*Variable-definition-**set***
*Procedure-definition-**set***
{ *Composite-state-graph* | *State-aggregation-node* }
[*Abstract*]

Composite-state-type-definition is extended from Basic SDL-2010 to include an optional *Composite-state-type-identifier*, a *State-entry-point-definition-**set***, a *State-exit-point-definition-**set***, (as an alternative to *Composite-state-graph*), a *State-aggregation-node* and optionally *Abstract*. The optional *Composite-state-type-identifier* of *Composite-state-type-definition* identifies the base type (super type) of a specialization. See clause 8.1.3 for more information on *Abstract*.

Concrete grammar

<composite state type diagram> ::=
<composite state type page>
| <state aggregation type page>

Basic SDL-2010 <composite state type diagram> is extended to allow <state aggregation type page> as an alternative to <composite state type page>.

<composite state type page> ::=
<frame symbol>
contains {
<composite state type heading> <page number area>
<composite state structure area> }
{ ***is connected to*** <gate on diagram> }*
{ ***is connected to*** <state connection point area> }*
[***is associated with*** <package use area>]

Basic SDL-2010 <composite state type page> is extended to allow <state connection point area> connections.

<state aggregation type page> ::=
<frame symbol>
contains {
<state aggregation type heading> <page number area>
<aggregation structure area> }
{ ***is connected to*** <gate on diagram> }*
{ ***is connected to*** <state connection point area> }*
[***is associated with*** <package use area>]

<composite state type heading> ::=
<type preamble>
state type [<qualifier>] <composite state type name>
[<formal context parameters>] [<virtuality constraint>]
[<specialization>]
[<agent formal parameters>]

<composite state type heading> is extended from Basic SDL-2010 to have optional

Documents Document Sharing Hub
Full document details: ITU-T Z.102 2021

<formal context parameters>, optional <formal context parameters> and optional <specialization>.

<state aggregation type heading> ::=
<type preamble>
state aggregation type [<qualifier>] <composite state type name>
[<formal context parameters>] [<virtuality constraint>]
[<specialization>]
[<agent formal parameters>]

8.1.2 Type expression

Type expression is extended from Basic SDL-2010, when the base type is a parameterized type to allow the type expression to denote new type formed by binding actual parameters to the base type. The use of type expression is extended from Basic SDL-2010 and it is used for defining one type in terms of another by specialization (see clause 8.4).

Concrete grammar

<type expression> ::=
<base type> [<actual context parameter list>]

<type expression> is extended from Basic SDL-2010 to allow <actual context parameter list>. It is valid to have <actual context parameter list> if and only if <base type> denotes a parameterized type. Context parameters are defined in clause 8.3. Outside a parameterized type, the parameterized type shall only be used by referring to its <identifier> in <type expression>.

Model

A <type expression> yields either the type identified by the identifier of <base type> in cases where there are no actual context parameters, or an anonymous type defined by applying the actual context parameters to the formal context parameters of the parameterized type denoted by the identifier of <base type>. The anonymous type definition is formed by:

1) copying the definition of the <base type> in the context where the construct using the <type expression> occurs and changing the name to an anonymous unique name;
2) replacing each occurrence of each <formal context parameter> name by the corresponding <actual context parameter> (if there is one) in the copy;
3) removing the <formal context parameter> from the <formal context parameter list> if there is a corresponding <actual context parameter>, and removing the <formal context parameters> if the <formal context parameter list> is empty;
4) replacing the <type expression> by a <type expression> with a <base type> that identifies the type with the anonymous unique name and no <actual context parameter list>.

NOTE 1 – Two textually identical <type expression>s with the <actual context parameter list> denote different types, because they have different anonymous names. To use the same type binding in different places, a type definition using the <type expression> should be used to name the <type expression>.

If some actual context parameters are omitted, the type is still parameterized.

In addition to fulfilling any static conditions on the definition denoted by the <base type>, usage of the <type expression> shall also fulfil any static condition on the resultant type.

NOTE 2 – The static properties on the usage of a <type expression> are possibly violated in the following cases, for example.

– When a scope unit has signal context parameters or timer context parameters, the condition that stimuli for a state have to be disjoint depends on the actual context parameters that will be used.
– When an output in a scope unit refers to a gate or a channel, which is not defined in the nearest enclosing type having gates, instantiation of that type results in an erroneous specification if there is no communication path to the gate.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

– When a procedure contains references to signal identifiers, remote variables and remote procedures, specialization of that procedure inside an agent results in an erroneous specification if the usage of such identifiers inside the procedure violates valid usage for the process.

– When state types are instantiated as parts of the same state aggregation, the resulting composite state is erroneous if two or more parts have the same signal in the input signal set.

– When a scope unit has an agent context parameter that is used in an output action, the existence of a possible communication path depends on which actual context parameter will be used.

– When a scope unit has a sort context parameter, application of an actual sort context parameter will result in an erroneous specification if a polymorphic assignment to a value is attempted in the specialized type.

– If a formal parameter of a procedure added in a specialization has the <parameter kind> in/out or out, a call in the supertype to a subtype (using this) will result in an omitted actual in/out or out parameter (that is, in an erroneous specification).

– If a formal procedure context parameter is defined with an at least constraint and the actual context parameter has added a parameter of <parameter kind> in/out or out, it is possible that a call of the formal procedure context parameter in the parameterized type results in an omitted actual in/out or out parameter (that is, in an erroneous specification).

If the scope unit contains <specialization> and any <actual context parameter>s are omitted in the <type expression>, the <formal context parameter>s are copied (while preserving their order) and inserted in front of the <formal context parameter>s (if any) of the scope unit. In place of omitted <actual context parameter>s, the names of corresponding <formal context parameter>s are inserted as <actual context parameter>s. These <actual context parameter>s now have the defining context in the current scope unit.

8.1.3 Abstract type

A type is an abstract type if its definition contains *Abstract*. An abstract type is either defined as abstract by the keyword **abstract** or because it has unbound context parameters.

Abstract grammar

Abstract :: {}

Abstract is part of the type definition. See clauses 8.1.1.1, 8.1.1.5, 9.4, 10.3 and 12.1.

Concrete grammar

<abstract> ::=
abstract

A type with unbound <formal context parameters> is also an abstract type.

NOTE – An abstract signal is allowed as a <signal constraint> in a <signal context parameter list>.

Semantics

An abstract type shall not be instantiated, therefore the *Abstract* property of a type prohibits instances of the type being used in the system, either from an explicit definition or because the type has unbound context parameters. However, instantiation of a subtype of an abstract data type is permitted, if the subtype is not itself abstract.

The *Abstract* property is not inherited: that is, whether a subtype is abstract depends on whether an explicit definition is given for the subtype, or if the subtype has unbound context parameters.

8.1.4 Gate

Gates are extended from Basic SDL-2010 to allow constraints to be specified on the types of agents that are connected (via a channel) to the gate, and allow the use of inherited gates in subtypes.

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

Concrete grammar

```
<gate definition> ::=
                    {           { <gate symbol 1> | <inherited gate symbol 1> }
                                is associated with { <gate> <signal list area> }set
                          |     { <gate symbol 2> | <inherited gate symbol 2> }
                                is associated with { <gate> <signal list area> <signal list area> }set
                    }     [ is connected to <endpoint constraint> ]
```

<gate definition> is extended compared with Basic SDL-2010 to allow <inherited gate symbol 1> and <inherited gate symbol 2> for inherited gates and <endpoint constraint>.

```
<endpoint constraint> ::=
                          { <block symbol> | <process symbol> | <state symbol> }
                          contains <textual endpoint constraint>

<textual endpoint constraint> ::=
                          [atleast] <identifier>

<inherited gate symbol 1> ::=
                          ---------->

<inherited gate symbol 2> ::=
                          <----------->
```

An <inherited gate symbol 1> or <inherited gate symbol 2> shall only appear in a subtype definition, and it shall be used in the subtype for the gate with the same <gate name> specified in the supertype.

<signal list area>s and any <endpoint constraint> associated with an <inherited gate symbol 1> or <inherited gate symbol 2> are additions to those of the gate definition in the supertype.

The <identifier> of the <textual endpoint constraint> of an <endpoint constraint> with a <block symbol> (<process symbol>, <state symbol>) shall denote the definition of a block type (process type, state type, respectively). If <textual endpoint constraint> is specified for the gate in the supertype, the <identifier> of an (added) <textual endpoint constraint> shall denote the same type or a subtype of the type denoted in the <textual endpoint constraint> of the supertype.

A channel connected to a gate shall be compatible with the endpoint constraint of the gate. A channel is compatible with this constraint if the other endpoint of the channel is an agent or state of the type denoted by <identifier> in the endpoint constraint or (if <textual endpoint constraint> contains **atleast**) a subtype of this type, and if the set of signals (if specified) on the channel is equal to, or is a subset of, the set of signals specified for the gate in the respective direction.

If the type denoted by <base type> in a <typebased block definition> or <typebased process definition> contains channels, the following rule applies: for each combination of a gate, a remote procedure (or remote variable), and the direction of the <signal list> of the gate defined by the type, the type shall contain at least one channel that – for the given direction – is connected to the frame at this gate and mentions the remote procedure or remote variable, respectively (or has no explicit <signal list area> associated if Recommendation [ITU-T Z.103] is being applied to allow the <signal list area> to be omitted).

Semantics

The use of gates in type definitions corresponds to the use of communication paths in the enclosing scope in (a set of) instance specifications.

8.2 Type references and operation references

See [ITU-T Z.101].

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

8.3 Context parameters

In order for a type definition to be used in different contexts, both within the same system specification and within different system specifications, it is allowed to parameterize types with context parameters. Context parameters are replaced by actual context parameters as defined in clause 8.1.2.

The following type definitions optionally have formal context parameters: system type, block type, process type, procedure, signal, composite state, interface and data type.

Context parameters optionally have constraints (that is, required properties any entity denoted by the corresponding actual identifier shall have). The context parameters have these properties inside the type.

Concrete grammar

```
<formal context parameters> ::=
                    <context parameters start> <formal context parameter list> <context parameters end>

<formal context parameter list> ::=
                    <formal context parameter> {<end> <formal context parameter> }*

<actual context parameter list> ::=
                    <context parameters start>
                    [<actual context parameter>] {, [<actual context parameter> ] }*
                    <context parameters end>

<actual context parameter> ::=
                    <identifier> | <constant primary>

<context parameters start> ::=
                    <less than sign>

<context parameters end> ::=
                    <greater than sign>

<formal context parameter> ::=
                    <agent type context parameter>
                |   <agent context parameter>
                |   <procedure context parameter>
                |   <remote procedure context parameter>
                |   <signal context parameter list>
                |   <variable context parameter list>
                |   <remotevariable context parameter list>
                |   <timer context parameter list>
                |   <synonym context parameter list>
                |   <sort context parameter>
                |   <compositestate type context parameter>
                |   <gate context parameter>
                |   <interface context parameter list>
```

The scope unit of a type definition with formal context parameters defines the names of the formal context parameters. These names are therefore visible in the definition of the type, and also in the definition of the formal context parameters.

An <actual context parameter> shall not be a <constant primary> unless it is for a synonym context parameter. A <constant primary> is a <primary> that is a valid <constant expression> (see clause 12.2.1 of [ITU-T Z.101]).

Formal context parameters are not allowed to be used as <base type> in <type expression> or in **atleast** constraints of <formal context parameters>.

Constraint is specified by a constraint specification, which is a <formal context parameter> with a constraint. A constraint specification introduces the entity of the formal context parameter followed by either a constraint signature or an **atleast** clause. A constraint signature introduces directly sufficient properties of the formal context parameter. An **atleast** clause denotes that the formal

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

context parameter shall be replaced by an actual context parameter, which is the same type or a subtype of the type identified in the **atleast** clause. Identifiers following the keyword **atleast** in this clause shall identify type definitions of the entity kind of the context parameter and shall be neither formal context parameters nor parameterized types.

A formal context parameter of a type shall be bound only to an actual context parameter of the same entity kind that meets the constraint of the formal parameter.

A context parameter using other context parameters in its constraint shall not be bound before the other parameters are bound, and if this means there is no possible order for binding the context parameters, the model is not valid SDL-2010.

It is allowed to omit trailing commas in the <actual context parameter list>.

Model

The formal context parameters of a type definition that is neither a subtype definition nor defined by binding formal context parameters in a <type expression> are the parameters specified in the <formal context parameters>.

Context parameters of a type are bound in the definition of a <type expression> to actual context parameters. In this binding, occurrences of formal context parameters inside the parameterized type are replaced by the actual parameters. During this binding of identifiers contained in <formal context parameter>s to definitions (that is, deriving their qualifier; see clause 6.6 of [ITU-T Z.101]), local definitions other than the <formal context parameters>s are ignored.

Parameterized types shall not be used as actual context parameters.

If a scope unit contains <specialization>, any omitted actual context parameter in the <specialization> is replaced by the corresponding <formal context parameter> of the <base type> in the <type expression>, and this <formal context parameter> becomes a formal context parameter of the scope unit.

8.3.1 Agent type context parameter

Concrete grammar

<agent type context parameter> ::=
{**process type** | **block type**} <agent type name> [<agent type constraint>]

<agent type constraint> ::=
atleast <agent type identifier> | <agent signature>

An actual agent type parameter shall be a subtype of the constraint agent type (**atleast** <agent type identifier>) with no addition of formal parameters to those of the constraint type, or it has to be compatible with the formal agent signature.

An agent type definition is compatible with the formal agent signature if it has the same kind and if the formal parameters of the agent type definition have the same sorts as the corresponding <sort>s of the <agent signature>.

8.3.2 Agent context parameter

Concrete grammar

<agent context parameter> ::=
{ **process** | **block** } <agent name> [<agent constraint>]

<agent constraint> ::=
{ **atleast** | <colon> } <agent type identifier> | <agent signature>

<agent signature> ::=
<sort list>
| [<end>] **fpar** <aggregation kind> <sort> {, <aggregation kind> <sort> }

Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021

NOTE – The two alternatives of <agent signature> are equivalent.

An actual agent parameter shall identify a block definition if **block** is specified in the <agent context parameter>; otherwise it shall identify a process definition.

In the case of **atleast** <agent type identifier> constraint, the actual agent type shall be the same as, or a subtype of, the constraint agent type with no addition of formal parameters to those of the constraint type.

In the case of a <colon> <agent type identifier> constraint, the actual agent type shall be the type denoted by <agent type identifier>.

In the case of an <agent signature> constraint, the actual agent shall be compatible with the <agent signature>. An agent definition is compatible with the <agent signature> if the formal parameters of the agent definition have the same aggregation kind and sorts as the corresponding elements of the <agent signature>, and both definitions have the same *Agent-kind*.

8.3.3 Procedure context parameter

Concrete grammar

```
<procedure context parameter> ::=
                    procedure <procedure name> <procedure constraint>

<procedure constraint> ::=
                    atleast <procedure identifier>
          |                 <procedure signature in constraint>

<procedure signature in constraint> ::=
                    <procedure signature>
          |         <legacy procedure signature>

<procedure signature> ::=
                    [ (<formal parameter> { , <formal parameter> }* ) ] [<result>]

<legacy procedure signature> ::=
                    [ <end> ]
          {                 [ returns <sort> ]
                    |       fpar <formal parameter> {, <formal parameter> }* [ <end> returns <sort> ] }
```

NOTE – The two alternatives of <procedure signature in constraint> are equivalent. The <sort> of <result> is equivalent to the <sort> after **returns**.

In the case of **atleast** <procedure identifier>, an actual procedure parameter shall identify a procedure definition that is the same as or specialization of the procedure of the constraint.

In the case of <procedure signature in constraint>, an actual procedure parameter shall identify a procedure definition compatible with the formal procedure signature given by <procedure signature in constraint>.

A procedure definition is compatible with the formal procedure signature if both have a result of the same <sort> or if neither returns a result, and:

a) the formal parameters of the procedure definition have the same sorts as the corresponding parameters of the signature each with the same <parameter kind>; or

b) each **in/out** and **out** parameter in the procedure definition has the same <sort identifier> or <syntype identifier> as the corresponding parameter of the signature.

8.3.4 Remote procedure context parameter

Concrete grammar

```
<remote procedure context parameter> ::=
                    remote procedure <procedure name> <procedure signature in constraint>
```


Documents
Document Sharing Hub
Full document details: ITU-T Z.102 2021