International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.100

(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 – Overview of SDL-2010

Recommendation ITU-T Z.100

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.

Recommendation ITU-T Z.100

Specification and Description Language – Overview of SDL-2010

Summary

Recommendation ITU-T Z.100 introduces the Specification and Description Language, intended for unambiguous specification and description of telecommunication systems. The scope of the Specification and Description Language is elaborated in clause 1. The ITU-T Z.100 series for SDL-2010 together form a reference manual for the language. The objective of this Recommendation is to provide an introductory overview to the language and the rest of the reference manual contained in the ITU-T Z.100 series for SDL-2010. The language introduced in this document is more fully defined in other Recommendations in the ITU-T Z.100 series for SDL-2010.

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. These foundations of the Specification and Description Language are elaborated in the respective main clauses of Recommendations ITU-T Z.101 to ITU-T Z.105 and ITU-T Z.107. A distinctive feature of the Specification and Description Language is the graphical representation. This Recommendation covers the conventions used to define the Specification and Description Language in the ITU-T Z.100 series, rules for conformance and guidance for maintenance of the language.

Applications

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 clause 1.2, but the Specification and Description Language is generally suitable for describing reactive systems. The range of application is from requirement description to implementation.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T Z.100	1984-10-19		11.1002/1000/2222
1.1	ITU-T Z.100 Annex A	1984-10-19		11.1002/1000/6664
1.2	ITU-T Z.100 Annex B	1984-10-19		11.1002/1000/6665
1.3	ITU-T Z.100 Annex C1	1984-10-19		11.1002/1000/6666
1.4	ITU-T Z.100 Annex C2	1984-10-19		11.1002/1000/6667
1.5	ITU-T Z.100 Annex D	1984-10-19		11.1002/1000/6668
2.0	ITU-T Z.100	1987-09-30	X	11.1002/1000/10954
2.1	ITU-T Z.100 Annex A	1988-11-25		11.1002/1000/6669
2.2	ITU-T Z.100 Annex B	1988-11-25		11.1002/1000/6670
2.3	ITU-T Z.100 Annex C1	1988-11-25		11.1002/1000/6671
2.4	ITU-T Z.100 Annex C2	1988-11-25		11.1002/1000/6672
2.5	ITU-T Z.100 Annex D	1988-11-25	X	11.1002/1000/3646
2.6	ITU-T Z.100 Annex E	1988-11-25		11.1002/1000/6673
2.7	ITU-T Z.100 Annex F1	1988-11-25	X	11.1002/1000/3647
2.8	ITU-T Z.100 Annex F2	1988-11-25	X	11.1002/1000/3648
2.9	ITU-T Z.100 Annex F3	1988-11-25	X	11.1002/1000/3649
3.0	ITU-T Z.100	1988-11-25		11.1002/1000/3153
3.1	ITU-T Z.100 Annex C	1993-03-12	X	11.1002/1000/3155
3.2	ITU-T Z.100 Annex D	1993-03-12	X	11.1002/1000/3156
3.3	ITU-T Z.100 Annex F1	1993-03-12	X	11.1002/1000/3157
3.4	ITU-T Z.100 Annex F2	1993-03-12	X	11.1002/1000/3158
3.5	ITU-T Z.100 Annex F3	1993-03-12	X	11.1002/1000/3159
3.6	ITU-T Z.100 App. I	1993-03-12	X	11.1002/1000/3160
3.7	ITU-T Z.100 App. II	1993-03-12	X	11.1002/1000/3161
4.0	ITU-T Z.100	1993-03-12	X	11.1002/1000/3154
4.1	ITU-T Z.100 (1993) Add. 1	1996-10-18	10	11.1002/1000/3917
5.0	ITU-T Z.100	1999-11-19	10	11.1002/1000/4764
5.1	ITU-T Z.100 (1999) Cor. 1	2001-10-29	17	11.1002/1000/5567
6.0	ITU-T Z.100	2002-08-06	17	11.1002/1000/6029
6.1	ITU-T Z.100 (2002) Amd. 1	2003-10-29	17	11.1002/1000/7091
6.2	ITU-T Z.100 (2002) Cor. 1	2004-08-29	17	11.1002/1000/356
7.0	ITU-T Z.100	2007-11-13	17	11.1002/1000/9262
8.0	ITU-T Z.100	2011-12-22	17	11.1002/1000/11387
8.1	ITU-T Z.100 Annex F1	2000-11-24	10	11.1002/1000/5239
8.2	ITU-T Z.100 Annex F2	2000-11-24	10	11.1002/1000/5576
8.3	ITU-T Z.100 Annex F3	2000-11-24	10	11.1002/1000/5577
8.4	ITU-T Z.100 Annex F1	2015-01-13	17	11.1002/1000/12354
8.5	ITU-T Z.100 Annex F2	2015-01-13	17	11.1002/1000/12355

8.6	ITU-T Z.100 Annex F3	2015-01-13	17	11.1002/1000/12356
9.0	ITU-T Z.100	2016-04-29	17	11.1002/1000/12846
9.1	ITU-T Z.100 Annex F1	2016-10-29	17	11.1002/1000/13040
9.2	ITU-T Z.100 Annex F2	2016-10-29	17	11.1002/1000/13041
9.3	ITU-T Z.100 Annex F3	2016-10-29	17	11.1002/1000/13042
9.4	ITU-T Z.100 Annex F1	2018-11-13	17	11.1002/1000/13732
9.5	ITU-T Z.100 Annex F2	2018-11-13	17	11.1002/1000/13733
9.6	ITU-T Z.100 Annex F3	2018-11-13	17	11.1002/1000/13734
10.0	ITU-T Z.100	2019-10-14	17	11.1002/1000/14048
10.1	ITU-T Z.100 Annex F1	2019-10-14	17	11.1002/1000/14049
10.2	ITU-T Z.100 Annex F2	2019-10-14	17	11.1002/1000/14050
10.3	ITU-T Z.100 Annex F3	2019-10-14	17	11.1002/1000/14051
11.0	ITU-T Z.100	2021-06-13	17	11.1002/1000/14670
11.2	ITU-T Z.100 Annex F2	2021-06-13	17	11.1002/1000/14702
11.3	ITU-T Z.100 Annex F3	2021-06-13	17	11.1002/1000/14703

Keywords

Specification and Description Language, SDL-2010, overview, conventions, grammars, type concept, presentation, tool compliance, system behaviour, system data description, system structuring.

* 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.

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/.

Table of Contents

Introduction

Status/Stability

This Recommendation is an introduction to the ITU-T Z.100 series of Recommendations for SDL-2010 that give the complete language reference manual for SDL-2010. The main text of this Recommendation is stable. Appendix I records the status of the Recommendation series, and should be updated as further studies are completed. The current language definition is based on wide user experience, recent additional user needs, clarifications and corrections. SDL-2010 as defined in this series of Recommendations should meet most user needs, and is based on a previous version called SDL-2000.

SDL-2000 contained a reference data type (**object type**) feature, but this had a number of complexities including dynamic binding, and tool support was lacking. It was therefore decided that this feature should be removed and further study took place leading to a further Recommendation in 2012 plus updates to other Recommendations in the ITU-T Z.100 series for a revised object-oriented data in SDL-2010 using a reference data type feature.

The main text is accompanied by appendices and annexes:

– Appendix I Status of ITU-T Z.100, related documents and Recommendations;
– Appendix II Guidelines for the maintenance of SDL-2010;
– Appendix III Evolution of the Specification and Description Language;
– Annex A Abstract syntax index;
– Annex B BNF syntax index;
– Annex C Compatibility;
– Annex D Data defined in the package Predefined;
– Annex E Reserved for examples.

The following Annex is published separately:

– Annex F Formal definition.

Annex F is the formal definition for the language and provides a more formal definition for SDL-2010 that provides more detail on issues not covered by the rest of the ITU-T Z.100 series for SDL-2010. If there is an inconsistency between Annex F and ITU-T Z.100 or other parts of the ITU-T Z.100 series for SDL-2010, there is an error in the ITU-T Z.100 series Recommendations and further study is needed to determine the correction.

The ITU-T Z.100 series has also an independently published supplement:

– ITU-T Z.Sup1: ITU-T Z.100 series – Supplement on SDL+ methodology: Use of ITU System Design Languages.

ITU-T Z.Sup1 is based on a methodology for an earlier version of the language. The methodology is still applicable because the earlier language features that are used all exist in SDL-2010. The current language has some additional features that are not fully exploited by ITU-T Z.Sup1.

Associated work

One method for usage within standards is described in Recommendation ITU-T Q.65. A recommended strategy for introducing a formal description technique like the Specification and Description Language in standards is available in Recommendation ITU-T Z.110. The use of the Specification and Description Language is also recommended in Recommendation ITU-T Z.450, Quality aspects of protocol-related Recommendations. For references to additional material on the Specification and Description Language, and information on industrial usage, see http://www.sdl-forum.org.

Background

Different versions of the Specification and Description Language have been recommended by ITU-T since 1976. The SDL-2010 version is a revision of SDL-2000, the last edition of which was published in 2007. SDL-2000 was initially published in Recommendation ITU-T Z.100 (1999) as a revision of Recommendation ITU-T Z.100 (1993) incorporating Addendum 1 to Recommendation ITU-T Z.100 (1996) and parts of Recommendation ITU-T Z.105 (1995). Recommendation ITU-T Z.100 (2002) was a technical update of Recommendation ITU-T Z.100 (1999) that incorporated a number of technical corrections and amendments, and without the textual phrase alternative syntax, which had been moved to Recommendation ITU-T Z.106 (2002).

Compared to the Specification and Description Language as defined in 1992, the versions defined in SDL-2000 and SDL-2010 are extended in the areas of object-oriented data, harmonization of a number of features to make the language simpler and features to enhance the usability of the Specification and Description Language with other languages such as ASN.1 and UML. Other minor modifications have been included. Though care has been taken not to invalidate existing documents using the Specification and Description Language as defined in 1992, it is possible some changes require some descriptions to be updated to use this version. Details on the evolution of the language are in Appendix III.

Recommendation ITU-T Z.100

Specification and Description Language – Overview of SDL-2010

1 Scope

The purpose of recommending the Specification and Description Language is to provide a language for unambiguous specification and description of the behaviour of telecommunication systems. The specifications and descriptions using the language are intended to be formal in the sense that it is possible to analyse and interpret them unambiguously.

The terms specification and description are used with the following meaning:

a) a specification of a system is the description of its required behaviour; and

b) a description of a system is the description of its actual behaviour (that is, its implementation).

A system specification, in a broad sense, is the specification of both the behaviour and a set of general parameters of the system. However, the Specification and Description Language is intended to specify the behavioural aspects of a system; the general parameters describing properties like capacity and weight have to be described using different techniques.

This Recommendation gives an overview of the series of Recommendations that define SDL-2010, defines terms, conventions including meta-languages, tool compliance and the basis of data for SDL-2010.

NOTE – Since there is no distinction between use for specification and its use for description, the term specification is used in the SDL-2010 Recommendations for both required behaviour and actual behaviour.

1.1 Objective

The general objectives when defining the Specification and Description Language have been to provide a language that:

a) is easy to learn, use and interpret;

b) provides unambiguous specification for ordering, tendering and design, while also allowing some issues to be left open;

c) is able to be extended to cover new developments;

d) is able to support several methodologies of system specification and design.

1.2 Application

The Recommendations for SDL-2010 provide the reference manual for the Specification and Description Language. This Recommendation provides an overview of the language and the conventions used to define the language. A methodology framework document, which gives examples of Specification and Description Language usage, is available as the Supplement to the Recommendation ITU-T Z.100 series originally produced in the study period 1992-1996.

The main area of application for the language is the specification of the behaviour of aspects of real-time systems, and the design of such systems. Applications in the field of telecommunications include:

a) call and connection processing (for example, call handling, telephony signalling, metering) in switching systems;

b) maintenance and fault treatment (for example, alarms, automatic fault clearance, routine tests) in general telecommunication systems;

c) system control (for example, overload control, modification and extension procedures);

d) operation and maintenance functions, network management;

e) data communication protocols;

f) telecommunication services.

The Specification and Description Language is, of course, usable for the functional specification of the behaviour of any object whose behaviour is specifiable using a discrete model; that is, where the object communicates with its environment by discrete messages.

The Specification and Description Language is a rich language and is usable for both high level informal (and/or formally incomplete) specifications, semi-formal and detailed specifications. The user chooses the appropriate parts of the Specification and Description Language for the intended level of communication and the environment in which the language is being used. Depending on the environment in which a specification is used, it is possible many aspects are left to the common understanding between the provider and the user of the specification.

Thus, the language is used for producing:

a) facility requirements;

b) system specifications;

c) ITU-T Recommendations, or other similar standards (international, regional or national);

d) system design specifications;

e) detailed specifications;

f) system design descriptions (both high level and detailed enough to directly produce implementations);

g) system testing descriptions (in particular in combination with Message Sequence Chart [MSC] and Testing and Test Control Notation [TTCN]).

The user organization is able to choose the appropriate level of application of SDL-2010.

1.3 System specification

A specification using the Specification and Description Language defines system behaviour in a stimulus/response fashion, assuming that both stimuli and responses are discrete and carry information. In particular, a system specification is seen as the sequence of responses to any given sequence of stimuli.

The system specification model is based on the concept of communicating extended finite state machines.

The Specification and Description Language also provides structuring concepts that facilitate the specification of large and/or complex systems. These constructs allow the partitioning of the system specification into manageable units that are capable of being handled and understood independently. It is possible to perform partitioning in a number of steps resulting in a hierarchical structure of units defining the system at different levels.

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

[ITU-T Z.102] Recommendation ITU-T Z.102 (2021), *Specification and Description Language – Comprehensive 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

There are numerous terms defined throughout this Recommendation and the rest of the ITU-T Z.100 series for SDL-2010 and a complete list of definitions in this clause or in each of these Recommendations would be a repetition of much of the text of the Recommendations. Therefore, only a few key terms are given in this clause.

This Recommendation defines the following terms:

3.1 agent: The term agent is used to denote a system, block or process that contains one or more extended finite state machines.

3.2 block: A block is an agent that contains one or more concurrent blocks or processes and is also permitted to contain an extended finite state machine that owns and handles data within the block.

3.3 body: A body is a state machine graph of an agent, procedure, composite state, or operation.

3.4 channel: A channel is a communication path between agents.

3.5 environment: The environment of the system is everything in the surroundings that communicates with the system in a Specification and Description Language-like way.

3.6 gate: A gate represents a connection point for communication with an agent type, and when the type is instantiated it determines the connection of the agent instance with other instances.

3.7 instance: An instance is an object created when a type is instantiated.

3.8 ITU-T Z.100 series for SDL-2010: This Recommendation and the associated Recommendations [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104], [ITU-T Z.105], [ITU-T Z.106], [ITU-T Z.107] and any further Recommendation subsequently added to this series.

3.9 pid: The term pid is used for the sort of data items that identify agent instances.

3.10 procedure: A procedure is an encapsulation of part of the behaviour of an agent, which is defined in one place but is able to be called from several places within the agent. Other agents are able to call a remote procedure.

3.11 process: A process is an agent that contains an extended finite state machine, and possibly contains other processes.

3.12 signal: The primary means of communication is by signals that are output by the sending agent and input by the receiving agent.

3.13 sort: A sort is a set of data items that have common properties.

3.14 state: An extended finite state machine of an agent is in a state if it is waiting for a stimulus.

3.15 stimulus: A stimulus is an event that is able to cause an agent that is in a state to enter a transition.

3.16 system: A system is the outermost agent that communicates with the environment.

3.17 timer: A timer is an item owned by an agent that causes a timer signal stimulus to occur at a specified time.

3.18 transition: A transition is a sequence of actions an agent performs until it enters a transition terminator such as the next state, a return from a composite state, a return from a procedure, or a decision on the subsequent transition.

3.19 type: A type is a definition that is used for the creation of instances, or is inherited and specialized to form other types. A parameterized type is a type that has parameters. When these parameters are given different actual parameters, different unparameterized types are defined that, when instantiated, give instances with different properties.

3.20 value: The term value is used for the class of data that is accessed directly. Values are allowed to be freely passed between agents.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

BNF	Backus-Naur Form
CIF	Common Interchange Format
IRV	International Reference Version of the International Reference Alphabet as defined in [ITU-T T.50]
MSC	Message Sequence Chart
SDL-88	Specification and Description Language as defined by Recommendation ITU-T Z.100 (1988)
SDL-92	Specification and Description Language as defined by Recommendation ITU-T Z.100 (1993) with Addendum 1 (1996)
SDL-2000	Specification and Description Language as defined by Recommendation ITU-T Z.100 (2007)
SDL-2010	Specification and Description Language as defined by the ITU-T Z.100 series for SDL-2010
TTCN	Testing and Test Control Notation
UCS	Universal Character Set of [ISO/IEC 10646]
UML	Unified Modelling Language

5 Conventions

The text of this clause defines the conventions used for describing the Specification and Description Language. The meta-languages and conventions introduced are solely introduced for the purpose of describing Specification and Description Language unambiguously.

The conventions of [ITU-T Z.111] apply to all the Recommendations in the ITU-T Z.100 series for SDL-2010.

5.1 Specification and Description Language grammars

In the ITU-T Z.100 series for SDL-2010 the *Abstract grammar* and *Concrete grammar* (see clause 5.3.2 below) define the Specification and Description Language. The syntax of the *Concrete grammar* is in some cases supplemented by a *Model* (see clause 5.3.2 below). A system specification only conforms to the language if it conforms to these grammars (see clause 5.1 of [ITU-T Z.111]). The way a system specification behaves is defined by *Semantics* (see clause 5.3.2 below).

A formal definition is provided which defines how to transform a system specification into the abstract syntax and defines how to interpret a specification, given in terms of the abstract grammar. The formal definition given in Annex F (published separately) is a comprehensive formal definition for SDL-2010.

5.2 Basic definitions

Some general concepts and conventions are used throughout the ITU-T Z.100 series for SDL-2010; their definitions are given in the following subclauses.

5.2.1 Definition, type and instance

In the ITU-T Z.100 series for SDL-2010, the concepts of type and instance and their relationship are fundamental. The schema and terminology defined below and shown in Figure 5-1 are used.

This subclause introduces the basic semantics of type definitions, instance definitions, parameterized type definitions, parameterization, binding of context parameters, specialization and instantiation.

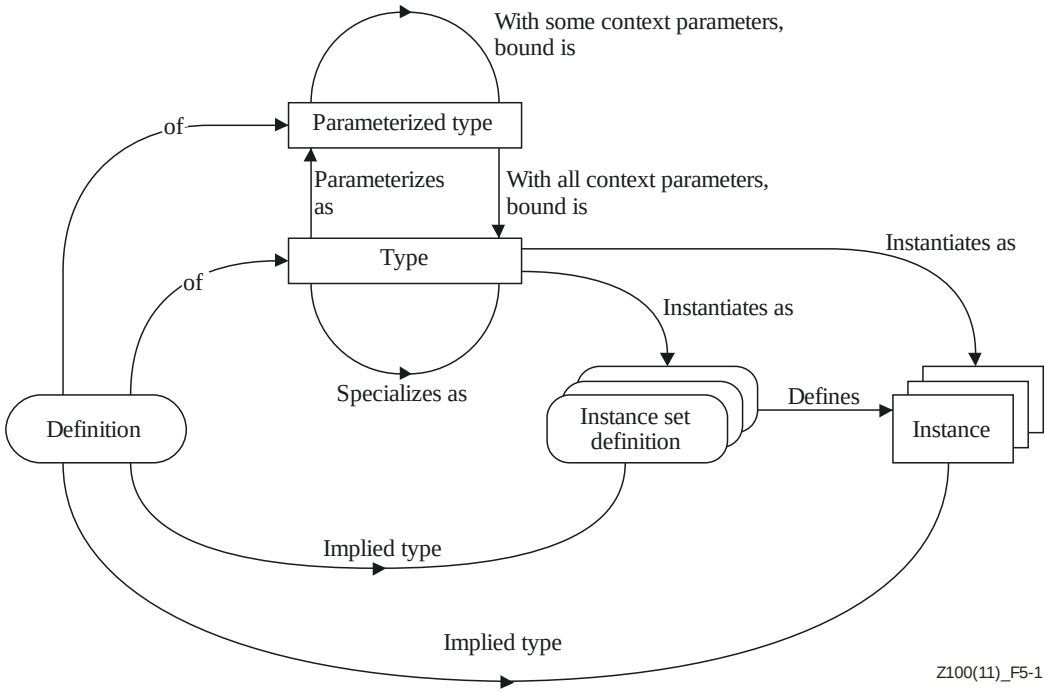


Figure 5-1 – The type concept

Definitions introduce named entities, which are types or instances with implied types or an instance set that defines the behaviour instances. A definition of a type defines all properties associated with that type. An example of an instance definition is a state definition. An example of a definition that is a type definition is a signal definition. An example of an instance set definition is a process definition. Block and process definitions introduce instance set definitions.

A type is allowed to be instantiated by any number of instances. An instance of a particular type has all the properties defined for that type. An example of a type is a procedure, which is instantiated by procedure calls.

A parameterized type is a type where some entities are represented as formal context parameters. A formal context parameter of a type definition has a constraint. The constraints allow static analysis of the parameterized type. Binding all the parameters of a parameterized type yields an ordinary type. An example of a parameterized type is a parameterized signal definition where one of the sorts conveyed by the signal is specified by a formal sort context parameter; this allows the parameter to be of different sorts in different contexts.

An instance is defined either directly or by the instantiation of a type. An example of an instance is a system instance, which is either defined by a system definition, or is an instantiation of a system type. However, where an instance is defined directly (for example, a system definition), this is actually an instantiation of the (anonymous) type for the direct definition.

Specialization allows one type, the subtype, to be based on another type, its supertype, by adding properties to those of the supertype or by redefining virtual properties of the supertype. A virtual property is allowed to be constrained in order to provide for analysis of general types.

Binding all context parameters of a parameterized type yields an unparameterized type. There is no supertype/subtype relationship between a parameterized type and the type derived from it.

NOTE – To avoid cumbersome text in the ITU-T Z.100 series for SDL-2010, the convention is used that the term "instance" is often omitted. For example, "a system is interpreted..." means "a system instance is interpreted...".

5.2.2 Environment

Systems that are specified in the Specification and Description Language behave according to the stimuli exchanged with the external world. This external world is called the environment of the system being specified.

It is assumed that there are one or more agent instances in the environment, and therefore stimuli flowing from the environment towards the system have associated identities of these agent instances. These agents have pids that are distinguishable from any other pid within the system (see clause D.2.16).

Although the behaviour of the environment is non-deterministic, it is assumed to obey the constraints given by the system specification.

5.2.3 Validity and errors

A system specification is a valid Specification and Description Language system specification only if it satisfies the syntactic rules and the static conditions defined in the ITU-T Z.100 series for SDL-2010.

If a valid Specification and Description Language specification is interpreted and a dynamic condition is violated, then an error occurs. Predefined exceptions (see clause D.2.20) will be raised when an error is encountered during the interpretation of a system. SDL-2010 does not define the handling of exceptions; therefore if an exception occurs the subsequent behaviour of the system cannot be derived from the specification.

For most cases where an exception might be raised (for example, a range check or incorrect indexing), it is possible to include actions to check before the error is encountered if the exception will be raised, and take appropriate action to avoid the error occurring. Static analysis or dynamic interpretation of a specification might also indicate that it is inevitable an exception is raised, leading to modification of the specification to avoid the situation.

5.3 Presentation style

The conventions of [ITU-T Z.111] apply.

5.3.1 Division of text

The conventions of [ITU-T Z.111] apply.

5.3.2 Titled enumeration items

Abstract grammar

The abstract grammar is specified in the form defined in clause 5.4.1 of [ITU-T Z.111]. The textual presentation of abstract syntax in clause 5.4.1.1 of [ITU-T Z.111] is used.

Concrete grammar

The concrete grammar is specified in the form defined in clause 5.4.2 of [ITU-T Z.111].

The metasymbol ***is followed by*** is used when the left-hand argument is followed by the right-hand argument, and this is shown in diagrams by a <flow line symbol> (see further description in clause 6.5 of [ITU-T Z.101]).

The right-hand argument of the metasymbol ***is associated with*** shall be closer to the left-hand argument than to any other graphical symbol. The syntactical elements of the right-hand argument shall be distinguishable from each other.

Except graphical symbols that are line symbols (such as <flow line symbol>, see the *Concrete grammar* description for the symbol), graphical symbol boundaries shall not overlay or cross. An exception to this rule applies for line symbols, which are allowed to cross each other. There is no logical association between line symbols that cross.

A line symbol consists of a line (solid or dashed) possibly with some additional decorations (typically an arrowhead) at one end or both ends of the line or on the line. The line of a line symbol consists of one or more joined straight-line segments. These segments should normally be horizontal or vertical.

Semantics

All instances have an identity property, but unless this is formed in some unusual way, this identity property is determined as defined by clause 6.6 of [ITU-T Z.101]. This is usually not mentioned as an identity property. Also, it has not been necessary to mention sub-components of definitions contained by the definition since the ownership of such sub-components is obvious from the abstract syntax. For example, it is obvious that a block type definition "has" enclosed processes and/or blocks.

Model

Some constructs are considered to be "derived concrete syntax" (or a shorthand notation) for other equivalent concrete syntax constructs. For example, omitting an input for a signal is derived concrete syntax for an input for that signal followed by a null transition back to the same state (see [ITU-T Z.103]).

Precise details of the order of transformation are found in Annex F.

5.4 Choice of grammar rules and rule names

The grammar given in the ITU-T Z.100 series for SDL-2010 has been written to aid the presentation in this Recommendation so that the rule names are meaningful in the context they are given and are readable in text. This means that there are a number of apparent ambiguities that are easily resolved by systematic rewriting of the syntax rules or the application of semantic rules.

6 Tool compliance

This clause defines the compliance for tools that claim to support the Specification and Description Language.

The validity of a specification is defined as in clause 5.2.3.

6.1 Definitions of valid tools

6.1.1 compliant SDL-2010 tool: A tool that detects non-compliance of a description with the ITU-T Z.100 series for SDL-2010. If the tool handles a superset notation, it is allowed to categorize non-compliance as a warning rather than a failure.

6.1.2 fully compliant SDL-2010 tool: A compliant SDL-2010 tool that supports the complete grammar defined by the ITU-T Z.100 series for SDL-2010.

6.1.3 valid basic SDL-2010 tool: A compliant SDL-2010 tool that supports the graphical grammar defined in this Recommendation in combination with [ITU-T Z.101].

6.1.4 valid SDL-2010 tool: A compliant SDL-2010 tool that supports the graphical grammar defined in the ITU-T Z.100 series for SDL-2010.

6.1.5 valid SDL-2010 with ASN.1 tool: A valid SDL-2010 tool that also supports ASN.1 as modules according to [ITU-T Z.105].

6.1.6 valid CIF SDL-2010 tool: A compliant SDL-2010 tool that supports the textual SDL-2010 grammar as defined in Level 0 CIF (see clause 5 of [ITU-T Z.106]), which (by definition) includes the semantics and some concrete syntax of other Recommendations in the ITU-T Z.100 series for SDL-2010.

6.1.7 valid CIF SDL-2010 with ASN.1 tool: A valid CIF SDL-2010 tool that also supports ASN.1 as modules according to [ITU-T Z.105].

6.2 Conformance

A conformance statement clearly identifying the language features and requirements not supported should accompany any tool that handles a subset of the language defined by the ITU-T Z.100 series for SDL-2010. If no conformance statement is provided, it shall be assumed that the tool is a fully compliant SDL-2010 tool. It is therefore preferable to supply a conformance statement; otherwise, any unsupported feature allows the tool to be rejected as not valid.

7 Allocation of features of SDL-2010 to Recommendations

The essential behaviour of a system defined using SDL-2010 depends on the extended finite state machine model of [ITU-T Z.101] coupled with the behaviour of expressions of [ITU-T Z.104]. The other Recommendations [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.105], [ITU-T Z.107] and [ITU-T Z.106] provide language features that (respectively): make the language more comprehensive, make the language easier and more practical to use, provide the full data model and action language (except object-oriented data), enable ASN.1 to be used, define object-oriented data, and define the interchange format. The following outlines the content of [ITU-T Z.101] to [ITU-T Z.107]. The content of Recommendations subsequently added to the ITU-T Z.100 series for SDL-2010 will be described in each Recommendation.

7.1 Basic SDL-2010 – [ITU-T Z.101]

[ITU-T Z.101] is the ITU-T Recommendation containing the part of the specification of SDL-2010 that covers core features such as agent (block, process) type diagrams containing agent instance structures with channels, diagrams for extended finite state machines and the associated semantics for these basic features. The character set used is International Reference Version (IRV) (see [ITU-T T.50]).

7.2 Comprehensive SDL-2010 – [ITU-T Z.102]

[ITU-T Z.102] is the ITU-T Recommendation containing the part of the specification of SDL-2010 that extends the semantics and syntax of the Basic SDL-2010 language in [ITU-T Z.101] to cover the full abstract grammar and the corresponding canonical concrete notation. This includes features such as continuous signals, enabling conditions, type inheritance and composite states. Also included are features for generic systems, macros and the handling of Universal Character Set (UCS) (see [ISO/IEC 10646]), though these do not need additional abstract grammar.

7.3 Shorthand notation and annotation in SDL-2010 – [ITU-T Z.103]

[ITU-T Z.103] is the ITU-T Recommendation containing the part of the specification of SDL-2010 that adds shorthand notations (such as asterisk state) that make the language easier to use and more concise, and various annotations that make models easier to understand (such as comments or create lines), but do not add to the formal semantics of the models. Models transform shorthand notations from the concrete syntax of [ITU-T Z.103] into concrete syntax of [ITU-T Z.102] or [ITU-T Z.101].

7.4 Data and action language in SDL-2010 – [ITU-T Z.104]

[ITU-T Z.104] is the ITU-T Recommendation containing the part of the specification of SDL-2010 that adds the data and action language used to define data types and expressions. In SDL-2010 the use of different concrete data notations is allowed, such as the SDL-2000 data notation or C, with bindings to the abstract grammar and the predefined data package.

The underlying data model is fundamental to behaviour and provides sorts of data such as Boolean and Integer that are used in other language features. For that reason this underlying model and an overview of predefined data sorts and constructs is given in Annex D of this Recommendation.

[ITU-T Z.104] does not define general reference or object data types or creation of data items other than as variables that are part of an agent, procedure or state instance. These issues are covered by [ITU-T Z.107].

7.5 SDL-2010 combined with ASN.1 modules – [ITU-T Z.105]

[ITU-T Z.105] provides a mapping for ASN.1 modules to features defined in the rest of the Specification and Description Language recommendations for SDL-2010, so that the ASN.1 modules define data items that are used with the rest of SDL-2010.

7.6 Common Interchange Format for SDL-2010 – [ITU-T Z.106]

[ITU-T Z.106] provides alternative textual syntax for the graphical syntax items defined in [ITU-T Z.101] to [ITU-T Z.105] that is used as a Common Interchange Format (CIF) between SDL-2010 tools. The basic level of CIF provides only a textual equivalent of graphical items. The full CIF is intended for the interchange of graphical SDL-2010 specifications (SDL-GR) so that the drawings are recognizably the same.

7.7 Object-oriented data in SDL-2010 – [ITU-T Z.107]

[ITU-T Z.107] defines the object-oriented data features of the Specification and Description Language building on the foundation of the data definitions and expressions defined in [ITU-T Z.104].

Annex A

Abstract syntax index

(This annex forms an integral part of this Recommendation.)

The abstract syntax index consists of the following table that lists the abstract grammar syntax rules of SDL-2010, where they are defined and redefined. Some abstract grammar syntax rules are defined in only in [ITU-T Z.102] or [ITU-T Z.104] as shown in the table below, in which case the column for [ITU-T Z.101] in the table below is shown blank. It is expected that users of the SDL-2010 Recommendations have access to machine-readable copies of the Recommendation texts and are therefore able to use computer software to locate the definitions and uses of the abstract grammar syntax rules. An abstract grammar syntax rule name should always be in italics, and the definition of a rule should start on a line in the original Microsoft Word text with the style "z.100 abs syntax 1st line".

Abstract syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.104	ITU-T Z.107
Abstract		definition		
Action-return-node	definition			
Activation-delay	definition			
Active-agents-expression	definition			
Active-expression	definition		redefinition	redefinition
Actual-parameters	definition			
Agent-definition	definition			
Agent-formal-parameter	definition			
Agent-identifier	definition			
Agent-instance			definition	
Agent-instance-pid-value			definition	
Agent-kind	definition			
Agent-name	definition			
Agent-qualifier	definition			
Agent-type-definition	definition	redefinition		
Agent-type-identifier	definition			
Agent-type-name	definition			
Agent-type-qualifier	definition			
Aggregation-kind	definition			redefinition
Alternative-expression	definition			
Any-decision		definition		
Any-expression			definition	
Argument	definition			
Assignment	definition			
Boolean-expression	definition			
Break-node		definition		
Call-node	definition	redefinition		

Abstract syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.104	ITU-T Z.107
Channel-definition	definition		redefinition	
Channel-endpoint	definition			
Channel-name	definition			
Channel-path	definition			
Closed-range	definition			
Composite-state-formal-parameter	definition			
Composite-state-graph	definition	redefinition		
Composite-state-type-definition	definition	redefinition		
Composite-state-type-identifier	definition			
Compound-node		definition		
Compound-node-name		definition		
Compound-node-qualifier		definition		
Condition-item	definition			
Conditional-expression	definition			
Connect-node	definition	redefinition		
Connection-definition		definition		
Connector-name	definition			
Consequence-expression	definition			
Constant-expression	definition		redefinition	redefinition
Continue-node		definition		
Continuous-expression		definition		
Continuous-signal		definition		
Create-request-node	definition			
Dash-nextstate	definition			
Data-type-definition	definition			
Data-type-identifier	definition			
Data-type-name	definition			
Data-type-qualifier	definition			
Decision-answer	definition			
Decision-body	definition			
Decision-node	definition	redefinition		
Decision-question	definition			
Decode-procedure-identifier			definition	
Decoding-expression			definition	
Default-initialization	definition			
Destination-gate	definition			
Destination-number	definition			
Direct-via	definition			
Dynamic-operation-signature				definition
Else-answer	definition			
Encoded-expression			definition	

Abstract syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.104	ITU-T Z.107
Encode-procedure-identifier			definition	
Encoding-expression			definition	
Encoding-path			definition	
Encoding-rules			definition	
Entry-connection-definition		definition		
Entry-procedure-definition		definition		
Equality-expression	definition			
Exit-connection-definition		definition		
Exit-procedure-definition		definition		
Expression	definition			
Finalization-node		definition		
First-operand	definition			
Formal-argument	definition			
Free-action	definition			
Gate-definition	definition		redefinition	
Gate-identifier	definition			
Gate-name	definition			
Graph-node	definition	redefinition		
Identifier	definition			
Imperative-expression	definition		redefinition	
In-choice			definition	
In-parameter	definition			
In-signal-identifier	definition			
Informal-text	definition			
Init-graph-node		definition		
Initial-number	definition			
Inner-entry-point		definition		
Inner-exit-point		definition		
Inout-parameter	definition			
Input-node	definition	redefinition	redefinition	
Instance-number			definition	
Interface-definition	definition			
Interface-name	definition			
Interface-qualifier	definition			
Join-node	definition			
Literal	definition			
Literal-identifier	definition			
Literal-name	definition			
Literal-natural	definition			
Literal-signature	definition			
Lower-bound	definition			

Abstract syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.104	ITU-T Z.107
Maximum-number	definition			
Name	definition			
Named-nextstate	definition			
Named-return-node		definition		
Named-start-node		definition		
Negative-equality-expression	definition			
Nextstate-node	definition			
Nextstate-parameters	definition	redefinition		
Now-expression	definition			
Null-literal-signature	definition			
Number-of-instances	definition			
Offspring-expression	definition			
Open-range	definition			
Operation-application	definition			
Operation-identifier	definition			
Operation-name	definition			
Operation-result	definition			
Operation-signature	definition			
Originating-gate	definition			
Out-parameter	definition			
Out-signal-identifier	definition			
Outer-entry-point		definition		
Outer-exit-point		definition		
Output-node	definition	redefinition	redefinition	
Package-definition	definition			
Package-name	definition			
Package-qualifier	definition			
Parameter	definition			
Parameter-aggregation	definition			
Parent-expression	definition			
Parent-sort-identifier	definition			
Path-item	definition			
Pid-expression	definition			
Positive-equality-expression	definition			
Priority-name		definition		
Procedure-definition	definition	redefinition		
Procedure-formal-parameter	definition			
Procedure-graph	definition			
Procedure-identifier	definition			
Procedure-name	definition			
Procedure-qualifier	definition			

Abstract syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.104	ITU-T Z.107
Procedure-start-node	definition			
Provided-expression		definition		
Qualifier	definition			
Range-check-expression	definition			
Range-condition	definition			
Reset-node	definition			
Result	definition			
Result-aggregation	definition			
Return-node	definition	redefinition		
Rules-identifier			definition	
Save-item	definition			
Save-signalset	definition			
Sdl-specification	definition			
Second-operand	definition			
Self-expression	definition			
Sender-expression	definition			
Set-node	definition			
Signal-definition	definition	redefinition		
Signal-destination	definition			
Signal-expression			definition	
Signal-identifier	definition			
Signal-name	definition			
Signal-parameter	definition			
Signal-priority	definition			
Signallist-expression			definition	
Size-constraint	definition			
Sort	definition			
Sort-identifier	definition			
Sort-reference-identifier	definition			
Spontaneous-transition		definition		
State-aggregation-node		definition		
State-entry-point-definition		definition		
State-entry-point-name		definition		
State-exit-point-definition		definition		
State-exit-point-name		definition		
State-expression			definition	
State-identifier	definition			
State-machine	definition			
State-name	definition			
State-node	definition	redefinition		
State-partition		definition		

Abstract syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.104	ITU-T Z.107
State-qualifier	definition			
State-start-node	definition			
State-timer		definition		
State-transition-graph	definition	redefinition		
State-type-name	definition			
State-type-qualifier	definition			
Static-operation-signature	definition			
Step-graph-node		definition		
Stop-node	definition			
Syntype-definition	definition			
Syntype-identifier	definition			
Syntype-name	definition			
Task-node	definition			
Terminator	definition	redefinition		
Time-expression	definition			
Timer-active-expression	definition			
Timer-default-initialization	definition			
Timer-definition	definition			
Timer-identifier	definition			
Timer-name	definition			
Timer-remaining-duration	definition			
Transition	definition			
Type-check-expression				definition
Type-coercion				definition
Value-data-type-definition	definition	redefinition		redefinition
Value-return-node	definition			
Value-returning-call-node	definition	redefinition		
Variable-access	definition			
Variable-definition	definition			
Variable-identifier	definition			
Variable-name	definition			
While-graph-node		definition		

Some abstract grammar syntax rules defined in [ITU-T Z.101] are extended by a redefinition in [ITU-T Z.102] or [ITU-T Z.104] or [ITU-T Z.107] (no rules are extended in [ITU-T Z.103]) as shown in the table above. For such rules the complete abstract grammar syntax of SDL-2010 is given by the redefined rule, which replaces the abbreviated rule given in [ITU-T Z.101]. Constraints and semantics given in [ITU-T Z.101] also apply to the redefined grammar, except if there is specific normative text otherwise. Constraints and semantics given in [ITU-T Z.102] or [ITU-T Z.104] or [ITU-T Z.107] for redefined rules are in addition to constraints and semantics in [ITU-T Z.101].

Annex B

BNF syntax index

(This annex forms an integral part of this Recommendation.)

The Backus-Naur Form (BNF) syntax index consists of the following table that lists the concrete grammar syntax rules of SDL-2010, where they are defined and redefined. For a rule defined in only in [ITU-T Z.104], the columns for [ITU-T Z.101], [ITU-T Z.102] and [ITU-T Z.103] in the table below are blank. Similarly, for a rule defined in [ITU-T Z.103] the columns for [ITU-T Z.101] and [ITU-T Z.102] are blank, and for a rule defined in [ITU-T Z.102], the columns for [ITU-T Z.101] is blank. It is expected that users of the SDL-2010 Recommendations have access to machine-readable copies of the Recommendation texts and are therefore able to use computer software to locate the definitions and uses of the concrete grammar syntax rules. A concrete grammar syntax rule name should always be of the form "<rule name>" and the definition of a rule should start "<rule name> ::=" at the start of a line that in the original Microsoft Word text has the style "z100 syntax 1st line". Subsequent lines defining a rule should use the user-defined Microsoft Word style "z100 syntax" except the last line of the rule definition which should use the style "z100 syntax last line".

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<abstract>		defined			
<action area>	defined	redefined			
<activation delay>	defined				
<active agents expression>	defined				
<active primary>	defined				
<actual context parameter list>		defined			
<actual context parameter>		defined			
<actual parameter list>	defined				
<actual parameter>	defined				
<actual parameters>	defined				
<agent additional heading>	defined	redefined			
<agent area>	defined	redefined	redefined		
<agent body area>			defined		
<agent constraint>		defined			
<agent context parameter>		defined			
<agent diagram>			defined		
<agent formal parameters>	defined		redefined		
<agent instance pid value>				defined	
<agent instance>				defined	
<agent instantiation>			defined		
<agent reference area>			defined		
<agent signature>		defined			
<agent structure area>	defined		redefined		
<agent text area>	defined	redefined	redefined		

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<agent type additional heading>	defined	redefined			
<agent type constraint>		defined			
<agent type context parameter>		defined			
<agent type diagram>	defined				
<agent type reference area>	defined		redefined		
<agent type reference>			defined		
<aggregation kind>	defined				redefined
<aggregation structure area>		defined			
<algorithm answer part>		defined			
<algorithm else part>		defined			
<alphanumeric>	defined				
<alternative expression>	defined				
<alternative question>		defined			
<alternative statement>		defined			
<ampersand>	defined				
<anchored sort>				defined	
<answer part>	defined				
<answer>	defined				
<any expression>				defined	
<apostrophe>	defined				
<argument>	defined				
<arguments>	defined				
<as channel>				defined	
<as gate>				defined	
<as interface>				defined	
<as signal>				defined	
<as signallist>				defined	
<assignment statement>	defined				
<assignment>	defined				
<asterisk connect list>			defined		
<asterisk input list>			defined		
<asterisk save list>			defined		
<asterisk state list>			defined		
<asterisk>	defined				
<base type>	defined				
<basic sort>	defined			redefined	
<basic state name>	defined				
<bit string>	defined				
<block diagram>			defined		
<block heading>			defined		

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<block page>			defined		
<block reference area>			defined		
<block reference>			defined		
<block symbol>	defined				
<block type diagram>	defined		redefined		
<block type heading>	defined				
<block type page>	defined				
<block type reference area>	defined		redefined		
<block type reference>			defined		
<block type symbol>	defined				
<break statement>		defined			
<call statement>		defined			
<channel definition area>	defined		redefined	redefined	
<channel symbol 1>	defined				
<channel symbol 2>	defined				
<character string>	defined				
<choice definition>	defined			redefined	
<choice list>	defined				
<choice of sort>	defined			redefined	
<circumflex accent>	defined				
<closed range>	defined				
<colon>	defined				
<comma>	defined				
<comment area>			defined		
<comment body>	defined				
<comment symbol>			defined		
<comment text>	defined				
<comment>	defined				
<commercial at>	defined				
<communication constraints>	defined	redefined			
<composite begin sign>	defined				
<composite end sign>	defined				
<composite primary>	defined				
<composite special>	defined				
<composite state body area>	defined	redefined			
<composite state diagram>			defined		
<composite state graph page>			defined		
<composite state heading>			defined		
<composite state list item>			defined		
<composite state name>	defined				

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<composite state reference area>			defined		
<composite state text area>	defined	redefined	redefined		
<composite state type constraint>		defined			
<composite state type diagram>	defined	redefined	redefined		
<composite state type heading>	defined	redefined			
<composite state type page>	defined	redefined			
<composite state type reference area>	defined		redefined		
<composite state type reference>			defined		
<composite state type signature>		defined			
<composite state type symbol>	defined				
<compositestate type context parameter>		defined			
<compound statement>		defined			
<concatenation sign>	defined				
<conditional expression>	defined				
<connect association area>	defined	redefined			
<connect list>		defined	redefined		
<connector name>	defined				
<consequence expression>	defined		redefined		
<constant expression>	defined				
<constant>	defined				
<constraint>	defined				
<context parameters end>		defined			
<context parameters start>		defined			
<continuous expression>		defined			
<continuous signal area>		defined			
<continuous signal association area>		defined			
<create body>	defined		redefined		
<create expression>				defined	
<create line area>			defined		
<create line endpoint area>			defined		
<create line symbol>			defined		
<create request area>	defined				
<create request symbol>	defined				
<create statement>		defined			
<dash nextstate>	defined				
<dashed association symbol>			defined		
<dashed block symbol>		defined			
<dashed line symbol>		defined			
<dashed process symbol>		defined			
<dashed state symbol>		defined			

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<data binding>				defined	
<data definition>	defined				
<data type constructor>	defined				
<data type definition body>	defined				
<data type definition>	defined			redefined	
<data type heading>	defined			redefined	
<data type specialization>				defined	redefined
<decimal digit>	defined				
<decision area>	defined	redefined			
<decision statement body>		defined			
<decision statement>		defined	redefined		
<decision symbol>	defined				
<decoding expression>				defined	
<default initialization>	defined			redefined	
<definition selection list>	defined				
<definition selection>	defined				
<definition>			defined	redefined	
<delaying channel symbol 1>	defined				
<delaying channel symbol 2>	defined				
<destination number>	defined				
<destination>	defined	redefined			
<diagram in package>	defined	redefined	redefined		
<diagram>	defined		redefined		
<dollar sign>	defined				
<else part>	defined				
<enabling condition area>		defined			
<enabling condition association area>		defined			
<enabling condition symbol>		defined			
<encoded input>				defined	
<encoded output>				defined	
<encoding expression>				defined	
<encoding path>				defined	
<encoding rules>				defined	
<end>	defined				
<endpoint constraint>		defined			
<entity in agent diagram>	defined				
<entity in composite state area>	defined				
<entity in data type>	defined			redefined	
<entity in interface>	defined			redefined	
<entity in operation>				defined	

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<entity in procedure>	defined	redefined	redefined		
<equality expression>	defined				
<equals sign>	defined				
<exclamation mark>	defined				
<exit transition area>	defined				
<export body>		defined			
<export statement>		defined			
<exported variable>		defined			
<exported variables of sort>		defined			
<exported>		defined			
<expression list>	defined				
<expression output>				defined	
<expression statement>			defined		
<expression>	defined				redefined
<expression0>	defined			redefined	
<extended primary>	defined				
<extended variable>	defined				
<external channel identifiers>			defined		
<external operation definition>				defined	
<external procedure definition>			defined		
<external synonym definition item>				defined	
<field default initialization>	defined				
<field list>	defined				
<field name>	defined				
<field number>				defined	
<field of kind>	defined				
<field primary>	defined			redefined	
<field sort>	defined				
<field variable>	defined			redefined	
<field>	defined				
<fields of sort>	defined			redefined	
<finalization statement>		defined			
<flow line symbol with arrowhead>	defined				
<flow line symbol without arrowhead>			defined		
<flow line symbol>	defined		redefined		
<formal context parameter list>		defined			
<formal context parameter>		defined			
<formal context parameters>		defined			
<formal name>		defined			
<formal operation parameters>	defined			redefined	

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<formal parameter>	defined				
<formal variable parameters>	defined				
<frame symbol>	defined				
<full stop>	defined				
<gate constraint>		defined			
<gate context parameter>		defined			
<gate definition>	defined	redefined	redefined	redefined	
<gate on diagram>	defined		redefined		
<gate property area>			defined		
<gate symbol 1>	defined				
<gate symbol 2>	defined				
<gate>	defined				
<general text character>	defined				
<graphical answer>	defined		redefined		
<grave accent>	defined				
<greater than or equals sign>	defined				
<greater than sign>	defined				
<hex string>	defined				
<history dash nextstate>	defined				
<history dash sign>	defined				
<hyphen>	defined				
<identifier>	defined				
<if statement>			defined		
<imperative expression>	defined			redefined	
<implies sign>	defined				
<import expression>		defined			
<imported procedure specification>			defined		
<imported variable specification>			defined		
<in choice>				defined	
<in connector area>	defined				
<in connector symbol>	defined				
<indexed primary>	defined				
<indexed variable>	defined				
<infix operation name>	defined				
<informal text>	defined				
<inherited agent definition>		defined			
<inherited block definition>		defined	redefined		
<inherited gate symbol 1>		defined			
<inherited gate symbol 2>		defined			
<inherited process definition>		defined	redefined		

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<inherited state machine>		defined			
<inherited state partition definition>		defined			
<initial number>	defined				
<inline data type definition>				defined	
<inline syntype definition>				defined	
<inner entry point>		defined			
<inner exit points>		defined			
<input area>	defined	redefined			
<input association area>	defined				
<input list>	defined		redefined	redefined	
<input symbol>	defined		redefined		
<integer name>	defined				
<interaction area>	defined		redefined		
<interface constraint>		defined			
<interface context parameter list>		defined			
<interface context parameter name>		defined			
<interface definition>	defined			redefined	
<interface gate definition>			defined	redefined	
<interface heading>	defined			redefined	
<interface procedure definition>				defined	
<interface specialization>				defined	
<interface use list>	defined				
<interface variable definition>				defined	
<internal input symbol>			defined		
<internal output symbol>			defined		
<internal procedure definition>			defined		
<internal synonym definition item>				defined	
<is assigned sign>	defined				
<keyword>	defined				
<left curly bracket>	defined				
<left parenthesis>	defined				
<left square bracket>	defined				
<legacy data inheritance>				defined	
<legacy data type definition>				defined	
<legacy external operator definition>				defined	
<legacy generator actual>				defined	
<legacy generators>				defined	
<legacy inheritance list>				defined	
<legacy inherited operator>				defined	
<legacy literal renaming>				defined	

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<legacy operator definition>				defined	
<legacy operator reference>				defined	
<legacy operator signature>				defined	
<legacy operator signatures>				defined	
<legacy procedure signature>		defined			
<legacy syntype definition>				defined	
<legacy task body>			defined		
<less than or equals sign>	defined				
<less than sign>	defined				
<letter>	defined				
<lexical unit>	defined				
<literal identifier>	defined				
<literal list>	defined			redefined	
<literal name>	defined				
<literal signature>	defined				
<literal>	defined				
<local variables of sort>		defined			
<loop alternative statement>		defined			
<loop answer part>		defined			
<loop body statement>		defined			
<loop break statement>		defined			
<loop clause>		defined			
<loop compound statement>		defined			
<loop consequence statement>			defined		
<loop continue statement>		defined			
<loop decision statement body>		defined			
<loop decision statement>		defined	redefined		
<loop else part>		defined			
<loop if statement>			defined		
<loop statement>		defined			
<loop statements>		defined			
<loop step>		defined			
<loop terminating statement>		defined			
<loop variable definition>		defined			
<loop variable indication identifier>		defined			
<loop variable indication>		defined			
<lower bound>	defined				
<lowercase letter>	defined				
<macro actual parameter>		defined			
<macro body>		defined			

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<macro call body>		defined			
<macro call>		defined			
<macro definition>		defined			
<macro formal parameter>		defined			
<macro formal parameters>		defined			
<macro parameter>		defined			
<macro symbol>			defined		
<mandatory field>	defined				
<maximum number>	defined				
<merge area>			defined		
<merge symbol>			defined		
<method application>				defined	
<method list>				defined	
<monadic operation name>	defined				
<name or number>	defined				
<name>	defined				
<named fields sort list>				defined	
<named number>	defined				
<nextstate area>	defined				
<nextstate body>	defined				
<nextstate body name>	defined				
<nextstate parameters>	defined	redefined			
<non terminating statement>	defined	redefined	redefined		
<non terminating statements>	defined	redefined			
<nondelaying channel symbol 1>	defined				
<nondelaying channel symbol 2>	defined				
<not asterisk or solidus>	defined				
<not equals sign>	defined				
<not number or solidus>	defined				
<note text>	defined				
<note>	defined				
<now expression>	defined				
<number of instances>	defined				
<number of pages>	defined				
<number sign>	defined				
<offspring expression>	defined				
<open range with operator>	defined				
<open range>	defined				
<operand>	defined				
<operand0>	defined				

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<operand1>	defined				
<operand2>	defined				
<operand3>	defined				
<operand4>	defined				
<operand5>	defined				
<operation application>	defined			redefined	
<operation body area>	defined				
<operation definition item>	defined			redefined	
<operation definition>				defined	
<operation definitions>	defined				
<operation diagram>	defined			redefined	
<operation heading>	defined			redefined	
<operation identifier>	defined				
<operation kind>	defined			redefined	
<operation name>	defined				
<operation page>	defined				
<operation preamble>				defined	redefined
<operation reference>	defined				
<operation result>	defined			redefined	
<operation signature>	defined			redefined	
<operation signatures>	defined			redefined	
<operation text area>	defined			redefined	
<operations>	defined				
<operator application>	defined				
<operator list>	defined				
<option area>		defined	redefined		
<option symbol>		defined			
<optional field>	defined				
<other character>	defined				
<other special>	defined				
<out connector area>	defined				
<out connector symbol>	defined				
<outer entry points>		defined			
<outer exit point>		defined			
<output area>	defined				
<output body item>	defined			redefined	
<output body>	defined		redefined		
<output statement>		defined			
<output symbol>	defined		redefined		
<package diagram>	defined		redefined		

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<package heading>	defined				
<package page>	defined				
<package public>	defined				
<package reference area>	defined		redefined		
<package reference>			defined		
<package symbol>	defined				
<package text area>	defined	redefined	redefined		
<package use area>	defined				
<package use clause>	defined			redefined	
<page number area>	defined				
<page number>	defined				
<parameter aggregation>	defined				
<parameter kind>	defined		redefined		
<parameters of sort>	defined				
<parent expression>	defined				
<parent sort identifier>	defined				
<path item>	defined				
<percent sign>	defined				
<pid expression>	defined				
<pid sort>	defined				
<plain input symbol>	defined				
<plain output symbol>	defined				
<plus sign>	defined				
<primary>	defined			redefined	
<priority clause>		defined	redefined		
<priority input area>		defined			
<priority input association area>		defined			
<priority input list>		defined	redefined		
<priority input symbol>		defined			
<priority name>		defined			
<priority stimulus>		defined	redefined		
<procedure body area>	defined				
<procedure call area>	defined				
<procedure call body>	defined	redefined			
<procedure call symbol>	defined				
<procedure constraint>		defined			
<procedure context parameter>		defined			
<procedure definition>			defined		
<procedure diagram>	defined				
<procedure formal parameters>	defined		redefined		

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<procedure heading>	defined	redefined			
<procedure page>	defined				
<procedure preamble>	defined	redefined			
<procedure reference area>	defined				
<procedure reference heading>	defined		redefined		
<procedure reference>			defined		
<procedure result>	defined		redefined		
<procedure signature in constraint>		defined			
<procedure signature>		defined			
<procedure start area>	defined	redefined			
<procedure start symbol>	defined				
<procedure symbol>	defined				
<procedure text area>	defined				
<process diagram>			defined		
<process heading>			defined		
<process page>			defined		
<process reference area>			defined		
<process reference>			defined		
<process symbol>	defined				
<process type diagram>	defined		redefined		
<process type heading>	defined				
<process type page>	defined				
<process type reference area>	defined		redefined		
<process type reference>			defined		
<process type symbol>	defined				
<provided expression>		defined			
<qualifier begin sign>	defined				
<qualifier end sign>	defined				
<qualifier>	defined				
<question mark>	defined				
<question>	defined				
<quotation mark>	defined				
<quoted operation name>	defined				
<range check constrained sort>	defined				
<range check expression>	defined				redefined
<range condition>	defined				
<range sign>	defined				
<range>	defined				
<real name>	defined				
<referenced definition>	defined		redefined		

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<remote procedure call area>		defined			
<remote procedure call body>		defined			
<remote procedure context parameter>		defined			
<remote procedure definition>		defined			
<remote variable definition>		defined			
<remote variables of sort>		defined			
<remotevariable context parameter list>		defined			
<remotevariable contextparameter names>		defined			
<rename list>				defined	
<rename pair literal name>				defined	
<rename pair operation name>				defined	
<rename pair>				defined	
<renaming>				defined	
<reset body>	defined		redefined		
<reset clause>	defined	redefined			
<reset statement>	defined				
<result aggregation>	defined				
<result sign>	defined				
<result>	defined				
<return area>	defined	redefined			
<return body>	defined				
<return statement>		defined			
<return symbol>	defined				
<reverse solidus>	defined				
<right curly bracket>	defined				
<right parenthesis>	defined				
<right square bracket>	defined				
<rules identifier>				defined	
<save area>	defined	redefined			
<save association area>	defined				
<save item>	defined				
<save list>	defined		redefined		
<save symbol>	defined				
<scope unit kind>	defined				
<sdl specification>	defined				
<select definition>		defined	redefined		
<selected entity kind>	defined	redefined			
<self expression>	defined				
<semicolon>	defined				
<sender expression>	defined				

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<set body>	defined		redefined		
<set clause>	defined				
<set statement>	defined				
<signal constraint>		defined			
<signal context parameter list>		defined			
<signal context parameter name>		defined			
<signal definition list>	defined				
<signal definition>	defined	redefined		redefined	
<signal expression>				defined	
<signal list area>	defined				
<signal list definition>			defined		
<signal list item>	defined	redefined	redefined		
<signal list symbol>	defined				
<signal list>	defined				
<signal priority>	defined				
<signal signature>		defined			
<signallist expression>				defined	
<simple expression>	defined				
<size constraint>	defined				
<solid association symbol>	defined				
<solidus>	defined				
<sort constraint>		defined			
<sort context parameter>		defined			
<sort list>	defined				
<sort signature>		defined			
<sort>	defined			redefined	
<space>	defined				
<special>	defined				
<specialization>		defined			
<spontaneous association area>		defined			
<spontaneous designator>		defined			
<spontaneous transition area>		defined			
<start area>	defined	redefined			
<start symbol>	defined				
<start timer area>			defined		
<start timer symbol>			defined		
<start>				defined	
<state aggregation body area>		defined			
<state aggregation heading>			defined		
<state aggregation page>			defined		

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<state aggregation type heading>		defined			
<state aggregation type page>		defined			
<state area>	defined	redefined			
<state connection point area>		defined			
<state connection point symbol 1>		defined			
<state connection point symbol 2>		defined			
<state entry point>		defined			
<state entry points>		defined			
<state exit point list>		defined	redefined		
<state exit point>		defined			
<state exit points>		defined			
<state expression>				defined	
<state list>	defined		redefined		
<state machine area>	defined	redefined	redefined		
<state partition area>		defined	redefined		
<state partition connection area>		defined			
<state symbol>	defined				
<state timer area>		defined			
<state timer association area>		defined			
<state timer>		defined			
<statement in loop>		defined			
<statement>	defined	redefined	redefined		
<statements>		defined	redefined		
<stimulus>	defined				
<stop statement>		defined			
<stop symbol>	defined				
<stop timer area>			defined		
<stop timer symbol>			defined		
<string name>	defined				
<structure definition>	defined			redefined	
<synonym constraint>		defined			
<synonym context parameter list>		defined			
<synonym context parameter name>		defined			
<synonym definition item>				defined	
<synonym definition>				defined	
<synonym>				defined	
<syntype definition data type>				defined	
<syntype definition syntype>	defined				
<syntype definition>	defined			redefined	
<syntype>	defined				

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<system diagram>			defined		
<system heading>			defined		
<system page>			defined		
<system specification>	defined		redefined		
<system type diagram>	defined		redefined		
<system type heading>	defined				
<system type page>	defined				
<system type reference area>	defined		redefined		
<system type reference>			defined		
<system type symbol>	defined				
<task area>	defined		redefined		
<task body>	defined		redefined		
<task symbol>	defined				
<terminating statement>		defined			
<terminator area>	defined	redefined	redefined		
<text extension area>			defined		
<text extension symbol>			defined		
<text symbol>	defined				
<text>			defined		
<textual endpoint constraint>		defined			
<tilde>	defined				
<timer active expression>	defined				
<timer communication constraint>		defined			
<timer constraint>		defined			
<timer context parameter list>		defined			
<timer context parameter name>		defined			
<timer default initialization>	defined				
<timer definition item>	defined				
<timer definition>	defined				
<timer remaining duration>	defined				
<transition area>	defined				
<transition option area>		defined			
<transition option symbol>		defined			
<transition string area>	defined				
<type coercion>					defined
<type expression>	defined	redefined			
<type preamble>	defined	redefined			
<typebased agent definition>	defined				
<typebased block definition>	defined				
<typebased block heading>	defined				

Concrete syntax rule name	ITU-T Z.101	ITU-T Z.102	ITU-T Z.103	ITU-T Z.104	ITU-T Z.107
<typebased composite state>	defined	redefined	redefined		
<typebased process definition>	defined				
<typebased process heading>	defined				
<typebased state partition definition>		defined			
<typebased state partition heading>		defined			
<typebased system definition>	defined				
<typebased system heading>	defined				
<underline>	defined				
<uppercase letter>	defined				
<valid input signal set>	defined				
<value returning procedure call>	defined			redefined	
<variable access>	defined	redefined		redefined	
<variable constraint>		defined			
<variable context parameter list>		defined			
<variable context parameter names>		defined			
<variable definition statement>		defined			
<variable definition>	defined	redefined			
<variable definitions>		defined			
<variable>	defined				
<variables of sort>	defined				
<vertical line>	defined				
<via path>	defined		redefined		
<virtuality constraint>		defined			
<virtuality>		defined			
<visibility>				defined	
<word>	defined				

Some concrete grammar syntax rules defined in [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103] or [ITU-T Z.104] are extended by a redefinition in [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104] or [ITU-T Z.107] as shown in the table above. Sometimes a redefined rule in [ITU-T Z.102] is further redefined. For redefined rules the complete concrete grammar syntax of SDL-2010 is given by redefinitions of the rule. A redefinition in [ITU-T Z.107] replaces the rule in [ITU-T Z.101] or [ITU-T Z.104] ([ITU-T Z.107] rules do not redefine rules in [ITU-T Z.102] or [ITU-T Z.103]). A redefinition in [ITU-T Z.104] replaces the rule in [ITU-T Z.101], [ITU-T Z.102] or [ITU-T Z.103]. A redefinition in [ITU-T Z.103] replaces the rule in [ITU-T Z.101] or [ITU-T Z.102] and redefinition in [ITU-T Z.102] replaces the rule in [ITU-T Z.101]. Constraints and semantics are cumulative so that those in [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103] or [ITU-T Z.104] also apply to any redefined grammar in [ITU-T Z.102], [ITU-T Z.103] or [ITU-T Z.104], except if there is specific normative text otherwise.

Any construct that is valid for a rule before redefinition should also be valid with the redefined rule. If this is not the case, there is an error in the ITU-T Z.100 series of SDL-2010 that needs to be corrected.

Annex C

Compatibility

(This annex forms an integral part of this Recommendation.)

SDL-2010 introduces some changes that could invalidate descriptions written for older versions of the Specification and Description Language that were available prior to SDL-2010 being approved. The intention is that SDL-2010 is compatible with most uses of the SDL-2000 versions as supported by tools. This annex documents how legacy descriptions are handled.

The Recommendation for SDL-2000 allowed many existing valid Specification and Description Language descriptions using SDL-92 to remain valid Specification and Description Language descriptions. Most (if not all) of these Specification and Description Language descriptions using SDL-92 should remain valid SDL-2010 descriptions.

Existing valid Specification and Description Language descriptions using SDL-2000 are not valid SDL-2010 descriptions if they use features of SDL-2000 deleted in SDL-2010, but it is expected the number of such cases is small because the deleted features (see clause III.4) have not been widely implemented or used.

In the concrete grammar of SDL-2010 there are a number of syntax rules introduced of the form <legacy …>. These rules define alternative syntax for features from older versions of the Specification and Description Language. New descriptions should avoid using such legacy syntax, but it is recognized that the use of legacy syntax is possibly impractical for existing descriptions, updating parts of existing descriptions or when the tool available only supports the legacy syntax.

Similarly, there are a number of concrete syntax rule alternatives containing the keywords **endnewtype**, **fpar**, **newtype** and **returns** that define alternative syntax from older versions of the Specification and Description Language. The concrete grammar defined by alternative syntax from older versions of the Specification and Description Language extends the notation allowed for SDL-2010 without extending the semantics. This grammar allows SDL-2010 tools to provide backwards compatibility for older descriptions and allows older tools to be used as tools for a subset of the SDL-2010 language. Where the alternative with the keyword occurs in Basic SDL-2010 [ITU-T Z.101] or Comprehensive SDL-2010 [ITU-T Z.102], this is the canonical form because it should be compatible with most tools, although an alternative without these keywords is preferred for new descriptions. The alternatives without the keywords are sometimes added in Shorthand SDL-2010 [ITU-T Z.103] or in [ITU-T Z.104].

Annex D

Data defined in the package Predefined

(This annex forms an integral part of this Recommendation.)

This annex provides an overview of the predefined data items that are defined in an implicitly used **package** Predefined, which is fully defined in [ITU-T Z.104]. The overview here provides an intuitive understanding of data and expressions for understanding [ITU-T Z.101], [ITU-T Z.102] and [ITU-T Z.103] without reference to [ITU-T Z.104]. The overview provided here should be consistent with [ITU-T Z.104], but any inconsistency should be assumed to be an error in this Annex that needs to be corrected.

D.1 Rules for "=" (equal), "/=" (not equal), comparison, data signatures and literals

The operators "=" (equal) and "/=" (not equal) are valid for every sort of data. The result is a Boolean.

The operators "<", "<=", ">", ">=" for comparisons are valid when the sort of data is based on ordered literals, for example, Character. If the name of the sort of data is S, it has the following operators:

```
"<" ( this S, this S )  -> Boolean;
">" ( this S, this S )  -> Boolean;
"<="( this S, this S )  -> Boolean;
">="( this S, this S )  -> Boolean;
first      -> this S;
last     -> this S;
succ( this S )    -> this S;
pred( this S )    -> this S;
num ( this S )    -> Natural;
```

NOTE – The quoted operators such as "<" are used as prefix operators, for example, in the expression "<"(a, b) but the quoted operator signs (in this case the less than sign) are also defined as infix operators so "<"(a, b) has the same meaning as a < b.

The keyword **this** before the name of the sort of data in signatures is relevant only when a sort of data is inherited.

Certain operators given below are able to raise an exception rather than return a result as indicated by **raise** and the name of the exception in the operator signature.

Where different sorts of data have literals that are lexically the same, the sort of data is usually determined by context. If this is not possible, the literal has to be qualified by the sort of data.

D.2 Package Predefined overview

The <<**package** Predefined>> is defined fully in [ITU-T Z.104]. This overview lists the sorts and the operations for each sort.

D.2.1 Boolean

```
literals true, false;
operators
     "not"( this Boolean )                -> this Boolean;
     "and"( this Boolean, this Boolean )  -> this Boolean;
     "or" ( this Boolean, this Boolean )  -> this Boolean;
     "xor"( this Boolean, this Boolean )  -> this Boolean;
     "=>" ( this Boolean, this Boolean )  -> this Boolean;
```

Boolean is used to represent true and false values. Often it is used as the result of a comparison. Boolean is used widely throughout the Specification and Description Language.

D.2.2 Character

```
literals
     NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
     BS,  HT,  LF,  VT,  FF,  CR,  SO,  SI,
     DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
     CAN, EM,  SUB, ESC, IS4, IS3, IS2, IS1,
     ' ', '!', '"', '#', '$', '%', '&', '''',
     '(', ')', '*', '+', ',', '-', '.', '/',
     '0', '1', '2', '3', '4', '5', '6', '7',
     '8', '9', ':', ';', '<', '=', '>', '?',
     '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
     'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
     'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
     'X', 'Y', 'Z', '[', '\', ']', '^', '_',
     '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
     'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
     'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
     'x', 'y', 'z', '{', '|', '}', '~', DEL;
/* '''' is an apostrophe, ' ' is a space, '~' is a tilde */
operators
     chr  ( Integer ) -> this Character;
```

The `Character` sort is used to represent characters of the International Reference Alphabet (Recommendation [ITU-T T.50]).

D.2.3 String

`String` is a parameterized sort of data that takes another sort of data as a parameter. For example, `String < Integer >` is a string of integers. In particular, the `Charstring` sort is defined as `String < Character >`.

In the following operators the sort of data given as the parameter is the `Itemsort`.

```
operators
     emptystring                                  -> this String;
     mkstring   ( Itemsort                     )  -> this String; /* same as Make */
     Make       ( Itemsort                     )  -> this String;
     length     ( this String                  )  -> Integer;
     first      ( this String                  )  -> Itemsort;
     last       ( this String                  )  -> Itemsort;
     "//"       ( this String, this String     )  -> this String; /* concatenation */
     Extract    ( this String, Integer         )  -> Itemsort raise InvalidIndex;
     Modify     ( this String, Integer, Itemsort) -> this String;
     substring  ( this String, Integer, Integer )  -> this String raise InvalidIndex;
     /* substring (s,i,j) gives a string of length j starting from the ith element */
     remove     ( this String, Integer, Integer )  -> this String;
     /* remove (s,i,j) gives a string with a substring of length j starting from
        the ith element removed */
```

Where the value of a `String` is needed, the notation

```
(. item .)
```

has the same meaning as

```
Make( item )
```

and makes a string of length 1 from the item value.

The notation where s is a string variable and i is an Integer

```
s [ i ]
```

has the same meaning as

```
Extract( s, i )
```

if it appears in an expression

and means there is an assignment to the element i of the string (using Modify) if it is the target of assignment.

D.2.4 Charstring

`Charstring` is `String` parameterized with `Character` and has the operators defined for `String`. Comparison operators and `num` are not defined for `Charstring`.

`Charstring` has literals that are defined by the regular expression

```
'''' ( (' ':'&') or '''''' or ('(': '~') )+ ''''
```

That is, the `Charstring` literal starts with an apostrophe, followed by one or more characters that are characters from space to ampersand or a pair of apostrophes (representing a single apostrophe) or a character from an open round bracket to a tilde and followed by an apostrophe. For example, 'aB%b 7cd'.

A double apostrophe not in a `Charstring` literal represents an empty `Charstring`.

'A' could be a `Character` or a `Charstring` depending on context. Non-printing characters are made into character strings by the use of the `Make` construct, so a character string for linefeed with carriage return and some prompt characters is constructed by

```
(. LF .) // (. CR .) // '+>'
```

D.2.5 Integer

A notation for an `Integer` value is a sequence of one or more of the numbers 0 to 9. Negative integers are represented by applying the unary "-" operator to a positive `Integer`. The ordering of integers is defined independently from the notation so that 0034 is ordered in the same place and has the same value as 034. The comparison operators are defined to give the usual mathematical integer ordering.

```
operators
     "-"    ( this Integer                   ) -> this Integer;
     "+"    ( this Integer, this Integer     ) -> this Integer;
     "-"    ( this Integer, this Integer     ) -> this Integer;
     "*"    ( this Integer, this Integer     ) -> this Integer;
     "/"    ( this Integer, this Integer     ) -> this Integer raise DivisionByZero;
     "mod"  ( this Integer, this Integer     ) -> this Integer raise DivisionByZero;
     "rem"  ( this Integer, this Integer     ) -> this Integer;
     "<"    ( this Integer, this Integer     ) -> Boolean;
     ">"    ( this Integer, this Integer     ) -> Boolean;
     "<="   ( this Integer, this Integer     ) -> Boolean;
     ">="   ( this Integer, this Integer     ) -> Boolean;
     power  ( this Integer, this Integer     ) -> this Integer;
     integer( << package Predefined>> Integer) -> this Integer;
     num    ( this Integer                   ) -> << package Predefined>> Integer;
```

`Integer` is used for mathematical integers with decimal 12, hex '0C'H, or binary '1100'B notation, where the examples given here all represent the same value (twelve).

There is no theoretical limit to the maximum size of an `Integer`, but in practice there will be some limit so some additional checks might be needed to ensure the limit is not reached. Using a **syntype** based on `Integer` with limits on the range of values instead of `Integer` provides such checks.

D.2.6 Natural

`Natural` is used when positive integers only are required. All operators will be the `Integer` operators, but the value is checked when a value is used as a parameter or assigned. A negative value will be an error.

D.2.7 Real

The notation for a `Real` value is a sequence of one or more of the numbers 0 to 9 followed by a decimal point (represented by a full stop) followed by one or more of the numbers 0 to 9. Examples are: 42.0; 10.3; 0.79 and .001.

The leading zeros before the decimal point do not have any meaning for the value, so that 001.4 has the same value as 1.4 and 000001.4. Similarly, trailing zeros after the decimal point do not change the value.

Negative values are represented by applying the unary "-" operator to a positive `Real`.

```
operators
     "-"   ( this Real             ) -> this Real;
     "+"   ( this Real, this Real  ) -> this Real;
     "-"   ( this Real, this Real  ) -> this Real;
     "*"   ( this Real, this Real  ) -> this Real;
     "/"   ( this Real, this Real  ) -> this Real raise DivisionByZero;
     "<"   ( this Real, this Real  ) -> Boolean;
     ">"   ( this Real, this Real  ) -> Boolean;
     "<="  ( this Real, this Real  ) -> Boolean;
     ">="  ( this Real, this Real  ) -> Boolean;
     float ( Integer               ) -> this Real;
     fix   ( this Real             ) -> Integer;
```

`Real` represents all numbers that are able to be represented as one integer divided by another (known as rational numbers). Numbers that cannot be represented in this way (irrational numbers – for example, the square root of 2) are not part of the `Real`. However, for practical engineering a sufficiently accurate approximation is usually used.

There is no theoretical limit to the maximum size of a `Real`, but in practice there will be some limit so some additional checks might be needed to ensure the limit is not reached. Similarly, there is no theoretical limit to the precision, so in theory values are infinitely precise but in practice there will be some limit so additional checks might be needed to allow for the precision of the implemented system. For example, the expression (1/9)*9=1.0 should always be true, but in an actual system it is possible the calculation of (1/9)*9 produces a value that is not precisely equal to 1.0.

D.2.8 The mapping called Array

`Array` is a mapping where a value of one sort of data is used as a key index to access element items of the mapping.

NOTE – The use of the name `Array` for this sort of data sometimes causes confusion, because in many computer languages the index of any data called an Array is limited to values of an ordered sort of data that has a finite number of elements such as `Character` or a bounded `Integer`. For an `Array` in the Specification and Description Language the index sort of data does not have to be ordered, nor does it need to be bounded. For example, the index is allowed to be `Integer` (which has no upper or lower bound), or a structured value. Vector is more similar to what is called an Array in other computer languages where the index is an `Integer` greater than zero with a maximum upper bound.

`Array` is a parameterized sort of data that takes two other sorts of data as parameters, in the following named the `Index` and the `Itemsort`.

```
operators
     Make                                  -> this Array ;
     Make   ( Itemsort                   ) -> this Array ;
     Modify ( this Array,Index,Itemsort  ) -> this Array ;
     Extract( this Array,Index           ) -> Itemsort raise InvalidIndex;
```

The Make notation produces an Array value as a result. If no parameter is given no element of the Array is initialized. Where a value of an Array is needed (determined by context), normally the notation

```
(. item .)
```

has the same meaning as

```
Make( item )
```

and makes an Array where every element is initialized to the item value. If the index is unbounded the Array is of a theoretically infinite size, but implementation is possible noting this item value once only for the Array and noting specifically {index, item value} pairs for elements that are given other item values. In this way only a finite amount of information is needed even for an Array of theoretically infinite size such as one indexed with Real.

The notation where a is an array variable and i is an index value

```
a [ i ]
```

has the same meaning as

```
Extract( a, i )
```

if it appears in an expression where it gives the value of the element

and means there is an assignment to the element i of the array (using Modify) if it is the target of assignment.

D.2.9 Vector

Vector is a parameterized sort of data that is an Array constrained to have an Integer index sort of data where the lower bound is 1 and the upper bound is the maximum index value given as the parameter of Vector.

D.2.10 Powerset

Powerset is a mathematical set. Every element item of a Powerset has the same sort of data. No item value appears more than once in the Powerset. If an item value is added to Powerset (using incl to include it), and the item value is already in the Powerset there is no change to the Powerset. There is no order to the elements of a Powerset.

```
operators
     empty                                      -> this Powerset;
     "in"   ( Itemsort, this Powerset       ) -> Boolean;       /* is member of          */
     incl   ( Itemsort, this Powerset       ) -> this Powerset; /* include item in set   */
     del    ( Itemsort, this Powerset       ) -> this Powerset; /* delete item from set  */
     "<"    ( this Powerset, this Powerset  ) -> Boolean;       /* is proper subset of   */
     ">"    ( this Powerset, this Powerset  ) -> Boolean;       /* is proper superset of */
     "<="   ( this Powerset, this Powerset  ) -> Boolean;       /* is subset of          */
     ">="   ( this Powerset, this Powerset  ) -> Boolean;       /* is superset of        */
     "and"  ( this Powerset, this Powerset  ) -> this Powerset; /* intersection of sets  */
     "or"   ( this Powerset, this Powerset  ) -> this Powerset; /* union of sets         */
     length ( this Powerset                 ) -> Integer;
     take   ( this Powerset                 ) -> Itemsort raise Empty;
```

The take operator removes and returns an arbitrary element item from the Powerset.

D.2.11 Duration

Duration is used for the values to be added to timers and as the result of the difference between Time values (see Time). The notation for Duration values is the same as for Real. Whether a value denotation is a Real or Duration (or Time) value is usually determined by context. The unit of Duration is the same as the unit of Time, and unless otherwise specified is 1 second.

```
operators
     "+"   ( this Duration, this Duration ) -> this Duration;
     "-"   ( this Duration                ) -> this Duration;
     "-"   ( this Duration, this Duration ) -> this Duration;
     ">"   ( this Duration, this Duration ) -> Boolean;
     "<"   ( this Duration, this Duration ) -> Boolean;
     ">="  ( this Duration, this Duration ) -> Boolean;
     "<="  ( this Duration, this Duration ) -> Boolean;
     "*"   ( this Duration, Real          ) -> this Duration;
     "*"   ( Real, this Duration          ) -> this Duration;
     "/"   ( this Duration, Real          ) -> Duration;
```

`Duration` values are allowed to be added and subtracted from one another, and multiplied and divided by `Real` values.

D.2.12 Time

Accessing the system clock returns a `Time` value. The origin of `Time` is system dependent. `Time` values are used to set the expiry time of timers.

The notation for `Time` values is the same as for `Real`. It is usually determined by context whether a value is a `Time` value or `Real` value or `Duration` value. The unit of `Time` is the same as the unit of `Duration`, and therefore unless otherwise specified is 1 second.

```
operators
     protected time  ( Duration   ) -> this Time;
     "<"   ( this Time, this Time  ) -> Boolean;
     "<="  ( this Time, this Time  ) -> Boolean;
     ">"   ( this Time, this Time  ) -> Boolean;
     ">="  ( this Time, this Time  ) -> Boolean;
     "+"   ( this Time, Duration) -> this Time;
     "+"   ( Duration, this Time) -> this Time;
     "-"   ( this Time, Duration) -> this Time;
     "-"   ( this Time, this Time  ) -> Duration;
```

A `Time` value that has a `Duration` added or subtracted from it gives another `Time` value. A `Time` value subtracted from another `Time` value gives a `Duration`.

D.2.13 Bag

`Bag` is an unordered collection of items. Every element item of a `Bag` has the same sort of data. By comparison with a `Powerset`, if an item value is added to Bag (using `incl` to include it), there is always a change to the `Bag` increasing the number of elements items that have the given value by one. Similarly, deleting an item of a particular value from a `Bag` (using `del`) reduces the number of elements items that have the given value by one (except if the count is already zero). Deleting an item of a given value is not an error if there are no items of that value in the `Bag`. There is no order to the elements of a `Bag`.

`Bag` is used to represent the SET OF construction of ASN.1.

```
operators
     empty                          -> this Bag;
     "in"   ( Itemsort, this Bag  ) -> Boolean;   /* is member of            */
     incl   ( Itemsort, this Bag  ) -> this Bag;  /* increase items of this value in Bag */
     del    ( Itemsort, this Bag  ) -> this Bag;  /* delete an item of this value from Bag
     */
     "<"    ( this Bag, this Bag  ) -> Boolean;   /* is proper subbag of     */
     ">"    ( this Bag, this Bag  ) -> Boolean;   /* is proper superbag of   */
     "<="   ( this Bag, this Bag  ) -> Boolean;   /* is subbag of            */
     ">="   ( this Bag, this Bag  ) -> Boolean;   /* is superbag of          */
     "and"  ( this Bag, this Bag  ) -> this Bag;  /* intersection of bags    */
     "or"   ( this Bag, this Bag  ) -> this Bag;  /* union of bags           */
     length ( this Bag            ) -> Integer;   /* number of items in Bag */
     count  ( Itemsort, this Bag  ) -> Integer;   /* number of this item value in Bag  */
     take   ( this Bag            ) -> Itemsort raise Empty;
```

The `take` operator removes and returns an arbitrary element item from the `Bag`.

D.2.14 Bit and Bitstring

`Bit` has the literals 0 and 1, and the same operators as `Boolean` plus operators for changing a `Bit` to an `Integer` and an `Integer` to a `Bit`.

```
operators
     "not" ( this Bit )                    -> this Boolean;
     "and" ( this Bit, this Bit )  -> this Bit;
     "or"  ( this Bit, this Bit )  -> this Bit;
     "xor" ( this Bit, this Bit )  -> this Bit;
     "=>"  ( this Bit, this Bit )  -> this Bit;
     num   ( this Bit )             -> Integer;
     bit   ( Integer            ) -> this Bit raise OutOfRange;
endvalue type Bit;
```

`Bitstring` is a string of each element of which is a `Bit`. To be compatible with ASN.1, the index of the first element is zero (unlike `Charstring` and other sorts of data based on `String` where indexing starts from 1). The values of `Bitstring` are denoted by the binary form '0101'B and/or the hexadecimal '12AF'H.

```
operators
     mkstring   (Bit                                  ) -> this Bitstring;
     Make       (Bit                                  ) -> this Bitstring;
     length     ( this Bitstring                      ) -> Integer;
     first      ( this Bitstring                      ) -> Bit;
     last       ( this Bitstring                      ) -> Bit;
     "//"       ( this Bitstring, this Bitstring      ) ->
                                    this Bitstring;/*concatenation*/
     Extract    ( this Bitstring, Integer             ) -> Bit raise InvalidIndex;
     Modify     ( this Bitstring, Integer, Bit        ) -> this Bitstring;
     substring  ( this Bitstring, Integer, Integer) -> this Bitstring raise InvalidIndex;
     /* substring (s,i,j) gives a string of length j starting from the ith element */
     remove     ( this Bitstring, Integer, Integer) -> this Bitstring;
     /* remove (s,i,j) gives a string with a substring of length j starting from
        the ith element removed */
/*The following operators are specific to Bitstrings*/
     "not"      ( this Bitstring                    ) -> this Bitstring;
     "and"      ( this Bitstring, this Bitstring    ) -> this Bitstring;
     "or"       ( this Bitstring, this Bitstring    ) -> this Bitstring;
     "xor"      ( this Bitstring, this Bitstring    ) -> this Bitstring;
     "=>"       ( this Bitstring, this Bitstring    ) -> this Bitstring
     num        ( this Bitstring                    ) -> Integer;
     bitstring  ( Integer                           ) -> this Bitstring raise OutOfRange;
     octet      ( Integer                           ) -> this Bitstring raise OutOfRange;
```

D.2.15 Octet and Octetstring

`Octet` is a `Bitstring` of exactly 8 bits, and therefore has the same notation as a `Bitstring` but with exactly 8 binary digits such as '10101010'B or 2 hexadecimal digits such as '1F'H.

`Octetstring` is a `String` of each element of which is an `Octet`. The index of the first element is one (like `Charstring` and other sorts of data based on `String`, but in contrast to `Bitstring` where indexing starts from zero). The values of `Octetstring` are multiples of 8 binary digits such as '0101010110101010'B or multiples of 2 hexadecimal digits such as 'FE12A7'H.

```
operators
     bitstring   ( this Octetstring ) -> Bitstring;
     octetstring ( Bitstring ) -> this Octetstring;
```

D.2.16 Process interface data (pid and Pid)

An interface defines a pid sort of data, which has elements that are identities of agents. The sort of data `Pid` is a supertype of all pid sorts of data. When a variable is declared to be of sort `Pid`, data items belonging to any pid sort are valid for assignment to that variable. Other pid sorts are constrained to reference only agents that offer the interface of the pid. Certain actions (such as creating a process instance) produce a `Pid` value. The notation `Null` denotes a reference that is not

associated with a value; that is the `Pid` value that is used when there is no instance to reference. Apart from equal and not equal, there are no language-defined operators for pid sorts of data. If there is an attempt to use `Null` as a reference the exception `InvalidReference` is raised.

D.2.17 Enumerated data

It is allowed to define a sort of data, the elements of which are named values. If the data sort is to be used in more than one place it is given a name. An example of introducing such a type named `enumabc` with values `a`, `b` and `c` is:

```
value type enumabc literals a, b, c;
```

D.2.18 Structure data

The keyword **struct** is used to introduce a composite structure sort of data that has named fields each of which is associated with a field sort of data. The fields have any sort of data, including a structure, an array, a vector or a string or a choice (see below). An example of a structure definition in a sort named `astruct` is:

```
value type astruct { struct f1 Integer; f2 Boolean; f3 Pid; }
```

The definition has make, extract and modify operators (similar to `String` and `Array`). Where a value for the above structure is needed (determined by context), the notation

```
(. ie1, be2, pe3 .)
```

is used to produce a structure value where ie1, be2 and pe3 are `Integer`, `Boolean` and `Pid` expressions respectively. This construct has the same meaning as

```
make( ie1, be2, pe3 )
```

If there is a variable vs declared with the structure above, the notations

```
vs.f1
vs.f2
vs.f3
```

are used to assign to, or access field `f1`, `f2` and `f3` respectively.

D.2.19 Choice

The keyword **choice** is used to introduce a composite sort of data that has named fields, only one of which is present at any time. If there is a variable defined with the choice sort of data, assigning to one of the fields means that all other fields lose the data value associated with them. An example of a choice definition in a sort named `achoice` is:

```
value type achoice { choice f1 Char; f2 Boolean; f3 Integer; }
```

If there is a variable `vc` declared with the choice above, the notations

```
vc.f1
vc.f2
vc.f3
```

are used to assign to, or access field `f1`, `f2` and `f3` respectively. To access a field to obtain a value of the field sort of data, the choice variable has to contain that field. If the choice variable does not contain the field when the access is made, the `UndefinedField` exception is raised. There is an operator `PresentExtract` with a parameter that is the choice sort the result of which is the name of the field that is present and is normally denoted by

```
vc.PresentExtract
```

For each field there is an operator with a parameter that is the choice sort, the result of which is a `Boolean` and has the value true if the choice value contains a value of that field. The name of the operator is the field name concatenated with Present, and the usual notation to apply the operator to a variable `vc` for a field `f1` is

```
vc.f1Present
```

`Choice` is used to represent the CHOICE construction of ASN.1.

D.2.20 Exceptions for language defined sorts of data

The following exceptions are mentioned in this annex:

```
OutOfRange,          /* A range check has failed. For example, assigning a negative
                        Integer value to a Natural, or applying the Bit operator or
                        Bitstring operator or Octetstring operator to negative Integer.
                        */
InvalidIndex,      /* A String or Array was accessed with an incorrect index. */
Empty;               /* No element could be returned. For example when applying the
                        take operator to a Powerset or Bag that is empty. */
DivisionByZero;      /* An Integer or Real division by zero was attempted. */
InvalidReference,    /* Null was used incorrectly. Wrong Pid for this signal. */
UndefinedField,      /* An undefined field was accessed. */
```

D.2.21 Support for ASN.1 character, symbol string and NULL types

The following items are defined in the **package** Predefined and support the combination of ASN.1 modules with SDL-2010: **syntype** NumericChar, **value type** NumericString, **syntype** PrintableChar, **value type** PrintableString, **syntype** TeletexChar, **syntype** VideotexChar, **value type** VideotexString, **syntype** IA5Char, **syntype** IA5String, **value type** GeneralChar, **value type** UniversalChar, **value type** UniversalCharString, **syntype** UTF8String, **value type** GeneralCharString, **syntype** GraphicChar, **syntype** VisibleChar, **value type** VisibleString, **syntype** BMPChar, **value type** BMPCharString and **value type** NULL.

Annex E

Reserved for examples

(This annex forms an integral part of this Recommendation.)

Annex F

Formal definition

(This annex forms an integral part of this Recommendation.)

Published separately. The status of Annex F is described in the Introduction and in Appendix I.

Appendix I

Status of ITU-T Z.100, related documents and Recommendations

(This appendix does not form an integral part of this Recommendation.)

This appendix contains a list of the status of Recommendations related to the Specification and Description Language issued by ITU-T. The list includes all parts of this Recommendation and of [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104], [ITU-T Z.105], [ITU-T Z.106], [ITU-T Z.107], [b-ITU-T Z.109], Recommendations subsequently added to the ITU-T Z.100 series for SDL-2010 and any related methodology documents. It also lists other relevant Recommendations such as Recommendations [b-ITU-T Z.110] and [ITU-T Z.111].

This list shall be updated by appropriate means (for example, a corrigendum) whenever changes to the Specification and Description Language are agreed and new Recommendations approved.

SDL-2010 is defined by or related to the following Recommendations approved by ITU-T listed below:

– Recommendation ITU-T Z.100.
– Annex A to Recommendation ITU-T Z.100.
– Annex B to Recommendation ITU-T Z.100.
– Annex C to Recommendation ITU-T Z.100.
– Annex D to Recommendation ITU-T Z.100.
– There were no specific plans at the time of approval for Annex E, but this is reserved for examples.
– Annex F to Recommendation ITU-T Z.100.
– Tools for the formal semantics reference model of SDL-2000 (ITU-T Specification and Description Language) are found at http://sourceforge.net/projects/sdlc (the files themselves are accessible either through CVS, or through the CVS web front end, at http://sdlc.cvs.sourceforge.net/viewvc/sdlc/).
– ITU-T Z.Sup1: ITU-T Z.100 series – Supplement on SDL+ methodology: Use of ITU System Design Languages.
– [ITU-T Z.101].
– [ITU-T Z.102].
– [ITU-T Z.103].
– [ITU-T Z.104].
– [ITU-T Z.105].
– [ITU-T Z.106].
– [ITU-T Z.107].
– [b-ITU-T Z.109].
– [b-ITU-T Z.110]. This is not part of the ITU-T Z.100-series of Recommendations for SDL-2010, but refers to ITU-T Z.100.
– [ITU-T Z.111]. This is not part of the ITU-T Z.100-series of Recommendations for SDL-2010, but included by reference.

Further information on the Specification and Description Language including information on books and other publications is available via: http://www.sdl-forum.org/.

Appendix II

Guidelines for the maintenance of SDL-2010

(This appendix does not form an integral part of this Recommendation.)

II.1 Maintenance of SDL-2010

This appendix describes the terminology and rules for maintenance of the Specification and Description Language agreed at the Study Group 10 meeting in November 1993, and the associated "change request procedure".

In the following text, references to Recommendation ITU-T Z.100 shall be considered to include annexes, appendices and supplements of this Recommendation, as well as any addenda, amendments, corrigenda or implementors' guides. This shall also apply for [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104], [ITU-T Z.105], [ITU-T Z.106], [ITU-T Z.107] and any Recommendation subsequently added to the ITU-T Z.100 series for SDL-2010 and [b-ITU-T Z.109].

II.1.1 Terminology

a) An *error* is an internal inconsistency within Recommendation ITU-T Z.100.

b) A *textual correction* is a change to text or diagrams of Recommendation ITU-T Z.100 that corrects clerical or typographical errors.

c) An *open item* is a concern identified but not resolved. An open item is identified either by a change request, or by agreement of the Study Group or Working Party.

d) A *deficiency* is an issue identified where the semantics of the Specification and Description Language is not (clearly) defined by Recommendation ITU-T Z.100.

e) A *clarification* is a change to the text or diagrams of Recommendation ITU-T Z.100 that clarifies previous text or diagrams that could be ambiguously understood without the clarification. The clarification should attempt to make Recommendation ITU-T Z.100 correspond to the semantics of the Specification and Description Language as understood by the Study Group or Working Party.

f) A *modification* is a change to the text or diagrams of Recommendation ITU-T Z.100 that changes the semantics of the Specification and Description Language.

g) A *decommitted feature* is a feature of the Specification and Description Language that is to be removed from the Specification and Description Language in the next revision of Recommendation ITU-T Z.100.

h) An *extension* is a new feature, which shall not change the semantics of features defined in Recommendation ITU-T Z.100.

II.1.2 Rules for maintenance

a) When an error or deficiency is detected in Recommendation ITU-T Z.100, it shall be corrected or clarified. The correction of an error should imply as small a change as possible. Error corrections and clarifications will be put into the master list of changes for Recommendation ITU-T Z.100 and come into effect immediately.

b) Except for error corrections and resolution of open items from the previous study period, modifications and extensions should only be considered as the result of a request for change that is supported by a substantial user community. A request for change should be followed by investigation by the Study Group or Working Party in collaboration with representatives of the user group, so that the need and benefit are clearly established and it is certain that an existing feature of the Specification and Description Language is unsuitable.

c) Modifications and extensions not resulting from error correction shall be widely publicized and the views of users and toolmakers canvassed before the change is adopted. Unless there are special circumstances requiring such changes to be implemented as soon as possible, such changes will not be recommended until Recommendation ITU-T Z.100 is revised.

d) Until a revised Recommendation ITU-T Z.100 is published, a master list of changes to Recommendation ITU-T Z.100 will be maintained covering Recommendation ITU-T Z.100 and all annexes except the formal definition. It is suggested the master list of changes is prepared as a draft version of an implementor's guide. Appendices, addenda, corrigenda, implementor's guides or supplements will be issued as decided by the Study Group. To ensure effective distribution of the master list of changes to Recommendation ITU-T Z.100, it will be published as COM Reports and by appropriate electronic means.

e) For deficiencies in Recommendation ITU-T Z.100, the formal definition should be consulted. This should lead to either a clarification or correction that is recorded in the master list of changes to Recommendation ITU-T Z.100. If there is an inconsistency between Recommendation ITU-T Z.100 and the formal definition, and it is decided that the formal definition is out of date or otherwise incorrect, it is permitted to document the inconsistency rather than update the formal definition.

II.1.3 Change request procedure

The change request procedure is designed to enable the Specification and Description Language users from within and outside ITU-T to ask questions about the precise meaning of Recommendation ITU-T Z.100, make suggestions for changes to the Specification and Description Language or Recommendation ITU-T Z.100, and to provide feedback on proposed changes to the Specification and Description Language. The Specification and Description Language experts' group shall publish proposed changes to the Specification and Description Language before they are implemented.

Requests for changes should either use the Change Request Form (see below) or provide the information listed by the form. The kind of request should be clearly indicated (error correction, clarification (or question), simplification, extension, modification or decommitted feature). It is also important that, for any change other than an error correction, the amount of user support for the request is indicated.

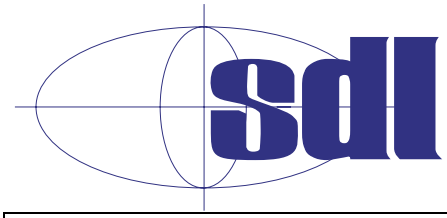
Meetings of the ITU-T Study Group responsible for Recommendation ITU-T Z.100 should treat all change requests. For corrections or clarifications, it is allowed that the changes are put on the list of corrections without consulting users. Otherwise, a list of open items shall be compiled. The information should be distributed to users:

– as ITU-T white contribution reports;

– by electronic mail to Specification and Description Language mailing lists (such as an ITU-T informal list for maintenance of the language, and sdlnews@sdl-forum.org);

– other means as agreed by the experts in the Study Group responsible for the Specification and Description Language.

Study group experts should determine the level of support and opposition for each change and evaluate reactions from users. A change will only be put on the accepted list of changes if there is substantial user support and no serious objections to the proposal from more than just a few users. Finally, all accepted changes will be incorporated into a revised Recommendation ITU-T Z.100. Users should be aware that until changes have been incorporated and approved by the Study Group responsible for Recommendation ITU-T Z.100, they are not recommended by ITU-T.



Change Request Form

Please supply the following details.		
Type of change:	❑ error correction	❑ clarification (or question)
	❑ simplification	❑ extension
	❑ modification	❑ decommission

Short summary of change request		
Short justification of the change request		
Is this view shared in your organization?	❑ yes	❑ no
Have you consulted other users?	❑ yes	❑ no
How many users do you represent?	❑ 1-5 ❑ 11-100	❑ 6-10 ❑ over 100
Your name and address		

Please attach further sheets with details if necessary.

SDL (ITU-T Z.100) Rapporteur, c/o ITU-T,
Place des Nations, CH-1211 Geneva 20, Switzerland.
Fax: +41 22 730 5853,
E-mail: SDL.rapporteur@itu.int.

Appendix III

Evolution of the Specification and Description Language

(This appendix does not form an integral part of this Recommendation.)

III.1 Versions of the Specification and Description Language

The Specification and Description Language was first Recommended in 1976. Since then it has had several major enhancements reflecting both changes in user needs and changes in techniques and tools, while retaining the state machine model of the original language.

It was first practical to make software tools for the language defined in Recommendation ITU-T Z.100 as published in the 1988 *Blue Book*. The language defined in the *Blue Book* is known as SDL-88, and the language defined in the next version of the Recommendation was called SDL-92. Every effort had been made to make SDL-92 a pure extension of SDL-88, without invalidating the syntax or changing the semantics of any existing SDL-88 usage. In addition, enhancements were only accepted based on need as supported by several ITU-T member-bodies. The SDL-92 language was quite successful, but some need for more significant change was established in the period 1996-2000, which resulted in the definition of SDL-2000, which was first published in 1999. Tools were already supporting some features of SDL-2000 at this time, but the legacy investment in SDL-92 of both tools and models coupled with some significant changes in business outlook meant that full migration of major tools to SDL-2000 was never achieved.

In the period 2000-2008 there was a need to integrate UML with ITU-T languages, which led to a UML profile for the Specification and Description Language in Recommendation ITU-T Z.109 in 2007. In the 2004 period to 2010 it was appropriate to review the language after a period of stability, in particular with respect to SDL-2000 features not supported by tools. At the same time SDL-2000 was maintained and incorporated changes were issued in 2007 consistent with the Recommendation ITU-T Z.109 of 2007. However, work on the profile in Recommendation ITU-T Z.109 (2007) also identified some desirable changes to SDL-2000 to better align the Specification and Description Language and UML. The changes made are documented in more detail in clause III.4.

III.2 Differences between SDL-88 and SDL-92

The major extensions were in the area of object orientation. While SDL-88 is object-based in its underlying model, some language constructs had been added to allow SDL-92 to more completely and uniformly support the object paradigm:

a) packages;

b) system, block, process and service types;

c) system, block, process and service (set of) instances based on types;

d) parameterization of types by means of context parameters;

e) specialization of types, and redefinition of virtual types and transitions.

The other extensions were: spontaneous transition, non-deterministic choice, internal input and output symbol for compatibility with existing diagrams, a non-deterministic imperative operator **any**, non-delaying channel, remote procedure call and value returning procedure, input of variable field, operator definition, combination with external data descriptions, extended addressing capabilities in output, free action in transition, continuous transitions in same state with same priority and m:n connections of channels and signal routes at structure boundaries. In addition, a number of minor relaxations to the syntax have been introduced.

In a few cases, changes were made to SDL-88 where the definition of SDL-88 was not consistent. It was possible to overcome the restrictions and changes introduced by an automatic translation procedure. This procedure was also necessary to convert an SDL-88 document into SDL-92 that contained names consisting of words that are keywords of SDL-92.

For the **output** construct, the semantics were simplified between SDL-88 and SDL-92, and this possibly invalidated some special usage of **output** (when no **to** clause is given and there exist several possible paths for the signal) in SDL-88 specifications. Also, some properties of the equality property of sorts were changed.

For the **export**/**import** construct, an optional remote variable definition was introduced, in order to align export of variables with the introduced export of procedures (remote procedure). This necessitated a change to any SDL-88 document that contained qualifiers in import expressions or introduced several imported names in the same scope with different sorts. In the (rare) cases where it was necessary to qualify import variables to resolve resolution by context, the change to make SDL-88 into SDL-92 is to introduce <remote variable definition>s and to qualify them with the identifier of the introduced remote variable name.

For the **view** construct, the view definition had been made local to the viewing process or service. This necessitated a change to SDL-88 documents that contained qualifiers in view definitions or in view expressions. To make SDL-88 into SDL-92 requires removal of these qualifiers. This did not change the semantics of the view expressions, since these are decided by their (unchanged) pid expressions.

The **service** construct was defined as a primitive concept, instead of being a shorthand form, without extending its properties. The use of service was not affected by this change, since it has been used anyway as if it were a primitive concept. The reason for the change is to simplify the language definition and align it with the actual use, and to reduce the number of restrictions on service, caused by the transformation rules in SDL-88. As a consequence of this change, the service signal route construct was deleted; signal routes could be used instead. This was only a minor conceptual change, and had no implications for concrete use (the syntax of SDL-88 service signal route and SDL-92 signal route were the same).

The **priority output** construct has been removed from the language. This construct is replaced by **output to self** with an automatic translation procedure.

Some of the definitions of the basic Specification and Description Language were extended considerably, e.g., **signal** definition. It should be noted that the extensions were optional, but were used for utilizing the power introduced by the object-oriented extensions, e.g., to use parameterization and specialization for signals.

Keywords of SDL-92 that are not keywords of SDL-88 are:

> **any, as, atleast, connection, endconnection, endoperator, endpackage, finalized, gate, interface, nodelay, noequality, none, package, redefined, remote, returns, this, use, virtual.**

III.3 Differences between SDL-92 and SDL-2000

A strategic decision was made to keep the language stable for the period 1992 to 1996, so that at the end of this period only a limited number of changes were made. These were published as Addendum 1 to Recommendation ITU-T Z.100 (1996) rather than updating the SDL-92 document. Although this version was sometimes called SDL-96, there were relatively few changes compared with those from SDL-88 to SDL-92. The changes were:

a) harmonizing signals with remote procedures and remote variables;

b) harmonizing channels and signal routes;

c) adding external procedures and operations;

d) allowing a block or process to be used as a system;

e) state expressions;

f) allowing packages on blocks and processes;

g) parameterless operators.

These were incorporated into Recommendation ITU-T Z.100, together with a number of other changes to produce a version known as SDL-2000 initially published as Recommendation ITU-T Z.100 (1999). In this Recommendation, the language defined by Recommendation ITU-T Z.100 (1993) with Addendum 1 to Recommendation ITU-T Z.100 (1996) is still called SDL-92. The 2002 version of SDL-2000 (the name was not changed) consolidated into Recommendation ITU-T Z.100 (1999) a number of technical changes made to correct errors or to improve the description of the language and to make a few minor extensions. Recommendation ITU-T Z.100 (2002) no longer included the alternative textual syntax of SDL-2000 that was instead defined in Recommendation ITU-T Z.106 (2002).

The advantages of language stability, which was maintained over the period from 1992 to 1996, began to be outweighed by the need to update the Specification and Description Language to support and better match other languages that are frequently used in combination with the Specification and Description Language. Also, improvements in tools and techniques had made it practical to generate software more directly from Specification and Description Language models, and incorporating better support for this use would provide further significant gains. While SDL-2000 is largely an upgrade of SDL-92, it was agreed that some incompatibility with SDL-92 was justified; otherwise, the resulting language would have been too large, too complex and too inconsistent. This subclause provides information about the changes from SDL-92 to SDL-2000.

Changes were made in a number of areas, with a focus on simplification of the language and adjustment made to newer application areas:

a) adjustment of syntactical conventions to other languages with which the Specification and Description Language is used;

b) harmonization of the concepts of system, block and process to be based on "agent", and merging of the concept of signal route into the concept channel;

c) interface descriptions;

d) exception handling;

e) support for textual notation of algorithms;

f) composite states;

g) replacement of the service construct with the state aggregation construct;

h) new model for data;

i) constructs to support the use of ASN.1 with the Specification and Description Language previously in Recommendation ITU-T Z.105 (1995).

Other changes were: nested packages, direct containment of blocks and processes in blocks, **out**-only parameters.

On the syntactic level, SDL-2000 is case-sensitive. Keywords are available in two spellings: all uppercase or all lowercase. Keywords of SDL-2000 that are not keywords of SDL-92 are:

> **abstract, aggregation, association, break, choice, composition, continue, endexceptionhandler, endmethod, endobject, endvalue, exception, exceptionhandler, handle, method, loop, object, onexception, ordered, private, protected, public, raise, value.**

The following keywords of SDL-92 are not keywords in SDL-2000:

all, axioms, constant, endgenerator, endrefinement, endservice, error, for, generator, literal, map, noequal, ordering, refinement, reveal, reverse, service, signalroute, view, viewed.

The following keywords of SDL-92 are keywords of SDL-2000 to support backwards compatibility:

endnewtype, fpar, imported, newtype, returns

A small number of constructs of SDL-92 were not available in SDL-2000: view expression, generators, block substructures, channel substructures, signal refinement, axiomatic definition of data, and macro diagrams. These constructs were rarely (if ever) used in SDL-92, and the overhead of keeping them in the language and tools did not justify their retention.

How most SDL-92 descriptions might be systematically transformed into SDL-2000 was given in Appendix III of Recommendation ITU-T Z.100 (1999) and subsequent versions up to Recommendation ITU-T Z.100 (2007), the last edition for SDL-2000.

III.4 Differences between SDL-2000 and SDL-2010

SDL-2010 supports Unicode for identifiers and annotations.

In SDL-2010 it is not allowed for a diagram to contain another diagram in the concrete syntax; instead the item that is included in another higher level item (for example, a process type in a block type) is shown as a reference symbol in the container diagram and the contained diagram is drawn separately. In SDL-2000 the change from a reference to an included item is theoretically done by transformations to a nested concrete syntax and then this concrete syntax is mapped to the abstract syntax. However, in reality such nesting of diagrams was not fully supported by tools, nor is it practical for any but the simplest systems. In SDL-2010 the change from references to the hierarchy is done by mapping referenced diagrams directly to the abstract grammar. In the concrete syntax the nested graphical form is no longer part of the language, with several simplifications to the concrete syntax because only one form of inclusion is provided. It is easy to redraw without nesting any existing diagrams that use nesting, and probably if a tool has been used to draw the diagrams, the tool will produce a version without nesting.

SDL-2010 does not include the UML-like concrete syntax of SDL-2000 for type references that look like UML class symbols. These had limited support in tools, but were a significant amount of additional concrete syntax in the language definition. The rationale at the time was that providing this syntax would aid the integration of SDL-2000 and UML descriptions. Since SDL-2000 was published, the UML profile for the Specification and Description Language [b-ITU-T Z.109] has been significantly improved, and it is now considered that tools provide the best way of integrating descriptions, so that additional complication of type reference syntax was not justified.

SDL-2010 does not include the optional <specification area>. Though such a mechanism is needed to collect all the diagrams together for a complete description, the approach taken is tool dependent and it does not need to be standardized. Related to the optional <specification area> and to the use of UML-like type references, the package dependency annotation is no longer part of SDL-2010.

SDL-2010 does not include the UML-like associations introduced in SDL-2000. These were not well supported and added nothing to the semantics of the language: they just made the language Recommendation more complex and difficult to understand. It is possible to present the information in auxiliary diagrams or in other annotation without defining a strict concrete syntax with constraints in the language Recommendations.

SDL-2010 does not include the **object** data type of SDL-2000. This was not widely supported by tools, and had a number of complications. However, the need for some of the benefits (in particular avoiding copying large data items, or multiple copying of data items) are still needed, and further study was progressed to provide these benefits as defined in [ITU-T Z.107].

The `nameclass` and `spelling` of SDL-2000 are not supported as user features in SDL-2010.

SDL-2010 includes predefined exceptions, but does not provide the exception handling mechanisms defined in SDL-2000. It is therefore well-defined when exceptions occur. If an exception occurs and is not handled, how the system subsequently behaves is not defined by the language (the same as SDL-2000). In most cases, language features or language-defined operations allow a check to be made before an exception might occur so that the situation is avoided. It is possible to replace user-defined exceptions by `Boolean` items. Exception handling pervaded many other features of the language, and removing it is a significant simplification.

Because exceptions are not handled, remote procedure call timers in SDL-2010 are managed in a way that differs from that of SDL-2000, as it is no longer possible to use an exception handler with the same name as the timer. Instead, if the timer expires, control is transferred to a connector that by default has the same name as the timer or is specifically named.

The SDL-2010 construct <agent instance pid value> enables initialization of values to denote the `Pid` values of the agent instances that exist when the system is initialized. Without such language defined initialization either the system has to be designed so that the `Pid` values of the agent instances are dynamically communicated between agents before normal handling of external signals commences, or some tool initialization of the values has to be used.

In SDL-2010, all timers of an agent are reset by the construct **reset** `*`, and all timers of an agent with the given timer name `timername` are reset by the construct **reset** `timername *`.

SDL-2010 has timer supervised states, where a **state timer** is optionally specified for a state and leads to a transition if the timer expires before another transition (other than an empty transition back to the same state) is taken. The timer is set on entering the state and reset for transition except an empty transition back to the same state.

The semantics of **synonym** is changed so that in principle it is a read-only variable, though the syntax is unchanged, and it is not expected that tools will need changing. The change was made so that a **synonym** is equivalent to a read-only variable in [b-ITU-T Z.109].

SDL-2010 allows the optional specification of a lower bound on the number of instances for an agent instance set. A *Stop-node* in an instance set that is already at the *Lower-bound* causes the exception `OutOfRange` to be raised. The number of active instances is returned by the integer built-in expression `active(this)` or `active(aid)` where aid is the name of an agent instance set definition. If *Lower-bound* is the same as *Maximum-number*, the number of instances is static and fixed when the agent instance set is created.

The signals that are available for input are placed in the input port of the agent instance to receive the signal. In SDL-2010, when a signal is placed in the input port, the retained information includes the identity of the gate on which it arrived at the SDL-2010 destination agent, which allows the taken transition to be determined by the gate as well as the signal identity by use of a **via** path. This enables the differentiation of instances of the same signal on different paths, rather than having to define a signal name for each path.

In SDL-2010, the values of unconsumed signals available in the input port can be examined through the **signallist** variable. The **signallist** variable has a string sort that is denoted **as signallist**, so that **signallist**`[n]` (where `n` is an integer) provides a choice value for an available signal. The string is ordered so that **signallist**`[1]` is the first available signal and the remaining string elements are in the availability order of the corresponding signals. The name of signal n is given by

signallist[n]!Present, and is used to access the values conveyed by the signal in the choice value **signallist**[n].

Priority input in SDL-2010 has multiple levels of input priority. A signal instance in the input port for an input with a higher priority is considered before those with lower priority.

In SDL-2010 a **signallist** definition has the same meaning as defining an **interface** that uses the listed signals.

In SDL-2010, it is possible to specify the delay between output of signal and the signal being available for consumption in the destination input port.

Normally, output of a signal results in one signal instance being conveyed to one of the reachable agent instances that has the signal in its valid input signal set. In SDL-2010, if **all** is specified in an output, then multiple signal instances are conveyed: one to each reachable agent instances that has the signal in its valid input signal set.

In SDL-2010, there are alternatives to specifying the name of signal to use in an output: an expression can be given where the sort of the expression is a choice sort that corresponds to a choice of signals that can be output; or if encoding rules are specified for a communication path and the output is directed via that path, an expression can be given that corresponds to the data type (Charstring, Octetstring or Bitstring) for that encoding. When a signal is input as an alternative to assigning each of the signal parameters to variables, the signal can be assigned to a variable with choice data type that corresponds to a choice of signals that can be input. The **signal** keyword denotes a variable that can hold any of the signals that can be received. The **signal** variable can be used in an input, the choice value of the **signal** variable can be accessed in expressions, and the **signal** variable can be used in an output to send a signal instance.

SDL-2010 allows an optional natural expression after the <agent identifier> of a <destination> to select a specific agent instance when there is more than one instance of the identified agent set, otherwise any existing instance of the set of agent instances is selected.

Bibliography

[b-ITU-T Z.109] Recommendation ITU-T Z.109 (2016), *Specification and Description Language – Unified modeling language profile for SDL-2010.*

[b-ITU-T Z.110] Recommendation ITU-T Z.110 (2008), *Criteria for use of formal description techniques by ITU-T.*

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	**Languages and general software aspects for telecommunication systems**

Printed in Switzerland
Geneva, 2021