International Telecommunication Union

ITU-T Implementer's guide

TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU

(September 2013)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

Specification and Description Language implementer's guide – Version 2.0.1

This page is intentionally blank.

Source

SDL implementer's guide version 2.0.1 was agreed on 4 September 2013 by ITU-T Study Group 17 (2013-2016)

CONTENTS

Page

Specification and Description Language implementer's guide – Version 2.0.1

1 Introduction

This Guide is a compilation of reported defects and their resolutions to the Specification and Description Language ITU-T Recommendations:

- Z.100, Z.101, Z.102, Z.103, Z.104, Z.105, Z.106, Z.107, Z.109, Z.111 and Z.119.

The Recommendations ITU-T Z.111 and Z.119 are included in the above list because they are essential normative references. Agreed changes to these documents that have not yet been issued in approved Recommendations are therefore listed here.

This Guide is intended to be an additional authoritative source of information for implementers to be read in conjunction with the Recommendations themselves.

This Guide itself is not an ITU-T Recommendation. However, it records agreed corrections to reported defects.

This Guide is for the SDL-2010 version of the language. The previous Guide version 1.0.2 was for the SDL-2000 version of the language and therefore did not include Recs. ITU-T Z.101, Z.102 and Z.103, and included the previous Z.107 (withdrawn in 2008). For convenience, Annex C includes changes to SDL-2000 versions of the relevant superseded Recommendations for users of old models and tools.

1.1 Scope of the Guide

The Guide records the resolution of defects and maintenance in the following categories as described in Rec. ITU-T Z.100 Appendix II Guidelines for maintenance of the Specification and Description Language:

- errors
- open items
- deficiencies
- clarifications
- modifications
- decommitted features
- extensions

NOTE: This Guide only addresses proposed additions, deletions, or modifications to the Recommendations that are strictly related to maintenance of the Specification and Description Language as described in the Z.100 series. Proposals for new features should be made in the normal way through contributions to ITU-T Study Group 17.

1.2 Approval of the Guide

This Guide is approved by ITU-T Study Group 17.

1.3 Distribution of the Guide

This Guide is available on-line at no charge from the ITU-T at (http://www.itu.int/rec/T-REC-Z.Imp100/en).

1.4 Contact

Any comments should be addressed to the ITU/TSB Secretariat for Study Group 17:

Martin Euchner
ITU/TSB
Place des Nations
CH-1211 Geneva 20
Switzerland

Tel: +41 22 730 5866
Fax: +41 22 730 5853
Email: tsbsg17@itu.int

2 Error reporting procedure

2.1 Submission of error reports and change requests

Any implementer of the Specification and Description Language defined in the ITU-T Z.100, Z.101, Z.102, Z.103, Z.104, Z.105, Z.106, Z.107 and Z.109 Recommendations is invited to submit a report using the form found in Appendix II of Z.100 and copied below in Annex A. The report should be submitted to the ITU-T Study Group 17 Secretariat (see clause 1.4). Each form should cover a single error ("error correction") or proposed change. Where the form reports an error, it is important that the form is completed accurately, especially the sections that relate to the base material against which the error report is being raised.

2.2 Resolution of errors

ITU-T Study Group 17 will address the submitted error. Following agreement on a resolution to the error, the proposed resolution will be approved using the appropriate procedures in ITU-T.

Please note that individual responses are not given specifically to those submitting reports, and that the procedure is not intended as a consulting service.

2.3 Documenting the Resolution of Defects

The following ITU-T Recommendations have errors or agreed changes. The defects and their resolutions are recorded in Annex B.

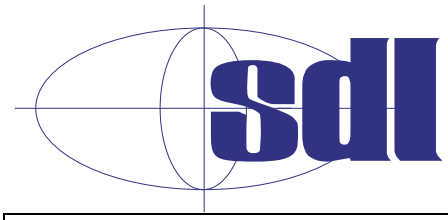
- None

The following ITU-T Recommendations had errors or agreed changes with respect to SDL-2000, but were replaced by revised Z.100 series Recommendations. The defects and their resolutions are recorded in Annex C.

- ITU-T Recommendation Z.100 (11/07) – one defect
- ITU-T Recommendation Z.106 (08/02) – one extension
- ITU-T Recommendation Z.109 (06/07) – eight defects, four clarifications

A number of errors or agreed changes with respect to ITU-T Recommendation Z.100 (08/02) were listed in Annex C of version 1.0.2 of this Guide, but were included in ITU-T Recommendation Z.100 (11/07) and therefore are not repeated here.

Annex A
Change Request Form



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.

Annex B

Master List of Changes

This is the master list of changes for the ITU-T Z.100 (SDL) series Recommendations for approved by the Working Party 5 (3) of Study Group 17 in 2013 (2009) according the rules for maintenance in Rec. ITU-T Z.100 itself.

History: The previous version of this list was published in version 2.0.0 of this document, which replaced versions 1.0.2, 1.0.1 and 1.0 of this document and the earlier document COM 17-TD 3250 [2001-2004] one of the documents of July 2004 WP C/17 meeting. COM 17-TD 3250 [2001-2004] records the history up to that point, and there seems to be no benefit repeating that historical information in this document.

In accordance with Appendix II to Recommendation ITU-T Z.100, the information in this document is distributed to users by various means including sdlnews@sdl-forum.org.

B.1 Objectives and scope

The purpose of this document is to record agreed changes to SDL Recommendations (ITU-T Z.100 to Z.109) and issues that require study and are therefore "open", or have been studied and a decision has been made that the issue is "closed": that is no further study should be undertaken.

The agreed changes come in two categories:

a) Correction of *errors* and *clarifications* (see definitions below and in Rec. ITU-T Z.100, Appendix II);

b) *Extensions* and *modifications* (see definitions below and in Rec. ITU-T Z.100, Appendix II).

The rules for maintenance in an Appendix to Rec. ITU-T Z.100, state that *errors* and *clarifications* published in the Master list of changes "come into effect immediately". Such changes should be published in a Corrigendum, Addendum or revision of the Recommendation as soon as is practical.

Modification and *extensions* imply some change to SDL. The rule in this case is "Unless there are special circumstances requiring such changes to be implemented as soon as possible, such changes will not be recommended until Rec. ITU-T Z.100 is revised."

B.2 Terminology

An *error* is an inconsistency in one or more Recommendations ITU-T Z.100 to Z.109.

A *textual correction* is a change in the text or diagrams of Recommendations that corrects clerical or typographical errors.

An *open item* is an issue identified but not resolved.

A *deficiency* is an issue identified where the semantics of SDL are not clearly defined in the Recommendations.

A *clarification* is a change to the text (or diagrams) in a Recommendation that does not (intentionally) change the meaning of SDL, but is intended to make the Recommendations less ambiguous or easier to understand.

A *modification* changes the semantics of SDL.

An *extension* is a new feature that does not change the semantics of SDL defined in the approved Recommendations for SDL.

B.3 Maintenance of Z.100 to Z.109

A Rec. ITU-T Z.100 Appendix documents the procedure to be followed for the maintenance of Recommendations ITU-T Z.100, Z.101, Z.102, Z.103, Z.104, Z.105, Z.106, Z.107 and Z.109. This procedure requires error corrections, proposed modifications and extensions to be widely publicized and a Master list of changes to be maintained. Clarifications or corrections for errors and deficiencies in the list of changes come into effect "immediately" (that is as soon as the Working Party or Study Group approves the list). Other changes take effect only when the relevant Recommendation is updated.

B.4 Z.100 changes

B.4.1 Defect correction – Annex A

See COM 17 TD 0571 [2013-2016] issue 2.

To correct a defect in equality expression in Z.101, *Positive-equality-expression* and *Negative-equality-expression* were added. These both need to be added to the table in Annex A as with "definition" in the column for Z.101.

B.5 Z.101 changes

B.5.1 Defect correction – 6.1 Lexical Rules <infix operation name>

See COM 17 TD 0571 [2013-2016] issue 1.

An <equals sign> (or <not equals sign>) should not be treated as an <infix operation name> in an expression but as part of an <equality expression>. The alternatives <equals sign> and <not equals sign> are removed from <infix operation name>, but are added to <quoted operation name> so that they can be used in the form "="(a,b). The revised rules are:

<quoted operation name> ::=
<quotation mark> <infix operation name> <quotation mark>
| <quotation mark> <monadic operation name> <quotation mark>
| <quotation mark> <equals sign> <quotation mark>
| <quotation mark> <not equals sign> <quotation mark>

<infix operation name> ::=

or	**xor**	**and**	**in**	**mod**	**rem**
<plus sign>	<hyphen>				
<asterisk>	<solidus>				
<greater than sign>	<less than sign>				
<less than or equals sign>	<greater than or equals sign>				
<concatenation sign>	<implies sign>				

B.5.2 Defect correction – 12.1.3 Operation signature, *Concrete grammar*

See COM 17 TD 0571 [2013-2016] issue 1.

As a consequence of changing the lexical rule <infix operation name>, the text in the penultimate paragraph before *Semantics*
'the user writes "(1 + 1) = 2" rather than having to use, for example, equal(add(1,1),2)'
needs to be changed to
'the user writes "(1 + 1) rem 2" rather than having to use, for example, rem(add(1,1),2)'

B.5.3 Extension – 12.1.6.2 Structure data types, *Concrete Grammar*

See COM 17 TD 0571 [2013-2016] 2.4.

The language is extended to thet the user is able to define a `Make` operator this is used in preference to the default `Make`.

b) if no operator named `Make` is present with a non-empty *Formal-argument* and an *Operation-result* that is the *Sort-reference-identifier* of the `S` structure sort list (an `S` structure result), an *Operation-signature* for a generic operator named `Make` with a non-empty *Formal-argument* list (where each item is the *Sort-reference-identifier* of the corresponding (in order) <field name>, and each formal parameter of the procedure identified by the *Operation-signature* has its *Parameter-aggregation* derived from the <aggregation kind> of the corresponding <field name>) and the `S` structure result with a *Result-aggregation* that is **PART**.

B.5.4 Defect correction – 12.1.7 Behaviour of operations, *Concrete Grammar*

See COM 17 TD 0571 [2013-2016] issue 1.

A constraint is put on the definition of quoted operators using <equals> and <not equals>. Add after the syntax for <operation heading> the following paragraph and NOTE:

If the <operation name> of the <operation heading> is a <quoted operation name> that is <quotation mark> <equals sign> <quotation mark> or is <quotation mark> <not equals sign> <quotation mark>, the number of <formal operation parameters> shall be less than one or greater than two.

NOTE - It is not allowed to define an operation with two parameters and an <operation name> that is <quotation mark> <equals sign> <quotation mark> or <quotation mark> <not equals sign> <quotation mark>. This avoids any ambiguity between an *Operation-application* and an *Equality-expression* for the generic equal operator.

B.5.5 Defect correction – 12.1.8.2 Constraint, *Concrete Grammar*

See COM 17 TD 0571 [2013-2016] issue 4.

The signature of the length operator for the size constraint is wrong. The name should be all lowercase, and the result should be Integer (rather than Natural).

Replace:

```
Length (in P ) -> <<package Predefined>>Natural;
```

By

```
length (in P ) -> <<package Predefined>>Integer;
```

In list items (i), (ii) and (iii) of the list item 3) if list item (c) replace every "Length" by "length".

B.5.6 Defect correction – 12.2.1 Operation signature, *Concrete grammar*

See COM 17 TD 0571 [2013-2016] issue 1.

As a consequence of changing the lexical rule <infix operation name>, the text in the paragraph starting "<operand>, <operand1>, <operand2>"
'the user writes "(1 + 1) = 2" rather than having to use, for example, equal(add(1,1),2)'
needs to be changed to
'the user writes "(1 + 1) rem 2" rather than having to use, for example, rem(add(1,1),2)'

B.5.7 Defect correction – 12.2.4 Equality expression

See COM 17 TD 0571 [2013-2016] issue 2.

There should be a distinction between a positive and negative equality expression in the abstract grammar, therefore the abstract grammar for *Equality-expression* is changed and *Positive-equality-expression* and *Negative-equality-expression* introduced.

The revised *Abstract grammar* syntax becomes:

Equality-expression	=	*Positive-equality-expression* | *Negative-equality-expression*
Positive-equality-expression	::	*First-operand* *Second-operand*
Negative-equality-expression	::	*First-operand* *Second-operand*
First-operand	=	*Expression*
Second-operand	=	*Expression*

In the *Concrete grammar* the paragraph after the syntax for <equality expression> is updated to:

An <equality expression> is legal concrete syntax only if the sort of one of its operands is sort compatible to the sort of the other operand. An <equality expression> using the <equals sign> represents a *Positive-equality-expression*. An <equality expression> using the <not equals sign> represents a *Negative-equality-expression*. The <operand2> represents a *First-operand*, and the <operand3> represents a *Second-operand*.

In the *Semantics*, "positive *Equality-expression*" is changed to "*Positive-equality-expression*" (4 times) and "negative *Equality-expression*" is changed to "*Negative-equality-expression*" (once).

B.5.8 Defect correction – 12.2.6 Operation Application, *Concrete Grammar*

See COM 17 TD 0571 [2013-2016] issue 1.

When the form "="(a,b) or "/="(a,b) is used for a <operation application>, this should be interpreted as an equity expression. Therefore add after the syntax for <operator application> the paragraph and NOTE:

If the <operation identifier> of the <operator application> is a <quoted operation name> that is <quotation mark> <equals sign> <quotation mark> or is <quotation mark> <not equals sign> <quotation mark>, and there are exactly two <actual parameters> where one <expression> of the <actual parameters> is sort compatible to the sort of the other <expression> of the <actual parameters>, the <operation application> represents an *Equality-expression* (see 12.2.4). The two <expression> items of the <actual parameters> represent *First-operand* and *Second-operand* of the *Equality-expression*. For an <equals sign> the *Equality-expression* is a *Positive-equality-expression*, and for <not equals sign> the *Equality-expression* is a *Negative-equality-expression*.

NOTE - *Equality-expression* uses the generic equal operator. It is not allowed to define an operation with two parameters and an <operation name> that is <quotation mark> <equals sign> <quotation mark> or <quotation mark> <not equals sign> <quotation mark>.

B.6 Z.102 changes

B.6.1 Defect correction – 11.14.1 Compound and loop statements, Abstract grammar

See COM 17 TD 0571 [2013-2016] section 2.1.

A *Finalization-node* should not end in a *Terminator* or *Decision*, and therefore should be a *Graph-node* rather than a *Transition*, because the interpretation of a *Finalization-node* is followed by followed by termination of the *Compound-node*. If the *Finalization-node* ended in a *Terminator* (or *Decision*) then this would follow the finalization statement and the termination of the *Compound-node* could not follow.

In the *Abstract grammar* change the abstract syntax for *Finalization-node* to:

Finalization-node	=	*Graph-node*

B.6.2 Defect correction – 11.14.1 Compound and loop statements, *Concrete grammar*

See COM 17 TD 0571 [2013-2016] issue 5.

Change the syntax for <loop variable definition> and the sentence following it to:

<loop variable definition> ::=
dcl <variable name> <aggregation kind> <sort> <is assigned sign> <expression>

Each <aggregation kind> <variable name> <sort> set in a <loop variable definition> represents the *Aggregation-kind*, *Variable-name* and *Sort-reference-identifier* of an element of the *Variable-definition-**set*** of the *Compound-node*.

B.6.3 Defect correction – 11.14.1 Compound and loop statements, Concrete grammar

See COM 17 TD 0571 [2013-2016] section 2.1.

A <finalization statement> should have a <compound statement> alternative, because an actual <statement> containing <method application> will be transformed into a <compound statement>, so <compound statement>has to be a valid replacement for the <statement>.

Change the syntax for <finalization statement> and the paragraph that follows it to:

<finalization statement> ::=
<statement>
| <compound statement>

A <finalization statement> represents the *Finalization-node* of the *While-graph-node* of the *Compound-node* of the <loop statement>. The <statement> or <compound statement> of the <finalization statement> represents the *Graph-node* of the *Finalization-node*.

B.6.4 Error correction – 12.1 Data definitions

Clerical error – no reference.

Correct error in the text
"extensions defined below for *Value-data-type-definition* and"
to
"extension defined below for *Value-data-type-definition*."

B.7 Z.103 changes

None.

B.8 Z.104 changes

B.8.1 Clarification – 12.2.6 Operation application, *Model*

See COM 17 TD 0571 [2013-2016] 2.3.

The *Model* for the case of a <method application> when the <expression> is not a <varibale> or **this** needed clarifying to make it clear what happens in the context of a statement and how to handle a <method application> not in a statement context.

Replace text for the *Model* by the following:

The concrete syntax form:
<expression> <full stop> <operation identifier> [<actual parameters>]

is derived concrete syntax for:
<operation identifier> *new-actual-parameters*

where *new-actual-parameters* is <actual parameters> containing only <expression>, if <actual parameters> was not present; otherwise, *new-actual-parameters* is obtained by inserting <expression> followed by a comma before the first optional expression in <actual parameters>.

If the <primary> of a <method application> is not a variable or **this**, one of the transformations below is applied before replacing the <full stop> as above, and there is an implicit assignment of the <primary> to an implicit variable with the sort of the first parameter of the operation (that is, the method sort) and this variable is used instead of the <primary>.

If the <method application> is in a statement, the following transformation is applied. The <statement> in which the <method application> occurs is replaced by a <left curly bracket> followed by a <variable definition> for the implicit variable, followed by an <assignment statement> assigning the <primary> to the implicit variable, followed by the original <statement> where the implicit variable replaces the <primary> in the <method application>. This is then either a <compound statement> or a <loop compound statement> depending on context.

If the <method application> is in an action that is not in a statement (for example, in the <question> of a <decision area>), the implicit variable is at the enclosing context and <task area> is placed in the flow immediately before the action. The task area contains the <assignment statement> and the implicit variable replaces the <primary> in the <method application>.

B.8.2 Defect correction – 14.7.1 Definition (for Real)

See COM 17 TD 0571 [2013-2016] issue 3.

The definition of Real did not define the literals with exponents and the value of these literals, even though the syntax for real literals on Z.101 includes exponents.

Change the literal name class definition to:

```
literals unordered nameclass
         (  ('0':'9')* ('0':'9') )
      or (  ('0':'9')* '.'('0':'9')+
            ('' or ('e' or 'E') ('' or '+' or '-' ) ('0':'9')* ('0':'9')) );
/*that is, integer notation, or decimal notation with an optional exponent */
```

Replace the line:

```
    for all i,j in Integer nameclass (
```

With

```
    for all i,j,k in Integer nameclass (
```

After the line

```
        spelling(s) == '.' // spelling(j) ==> r == (float(i) + s) / 10;
```

Insert the lines

```
/* */
        spelling(r) == spelling(i) // '.' // spelling(j) // 'e' // spelling (k),
        spelling(s) == spelling(i) // '.' // spelling(j) ==> r == s * float(power(10,k));
/* */
        spelling(r) == spelling(i) // '.' // spelling(j) // 'E' // spelling (k),
        spelling(s) == spelling(i) // '.' // spelling(j) ==> r == s * float(power(10,k));
/* */
        spelling(r) == spelling(i) // '.' // spelling(j) // 'e+' // spelling (k),
        spelling(s) == spelling(i) // '.' // spelling(j) ==> r == s * float(power(10,k));
/* */
        spelling(r) == spelling(i) // '.' // spelling(j) // 'E+' // spelling (k),
        spelling(s) == spelling(i) // '.' // spelling(j) ==> r == s * float(power(10,k));
/* */
        spelling(r) == spelling(i) // '.' // spelling(j) // 'e-' // spelling (k),
        spelling(s) == spelling(i) // '.' // spelling(j) ==> r == s / float(power(10,k));
/* */
        spelling(r) == spelling(i) // '.' // spelling(j) // 'E-' // spelling (k),
        spelling(s) == spelling(i) // '.' // spelling(j) ==> r == s / float(power(10,k));
```

B.8.3 Defect correction – 14.14.1 Definition (for Bitstring)

See COM 17 TD 0571 [2013-2016] 2.2.

Bitstring is corrected so the the least significant bit is the rightmost bit, and when converting to octet, any bitstring shorter than 8 bits is extended with zeros to the right.

Replace

```
/*Definition of 'xxxxx'B literals */
     for all s in Bitstring (
     for all b in Bit (
     for all i in Integer (
     <<type Bitstring>>num (''B)            == 0;
     <<type Bitstring>>bitstring (0)      == '0'B;
     <<type Bitstring>>bitstring (1)      == '1'B;
     num (s // mkstring (b))              == num (b) + 2 * num (s);
     i > 1                ==> bitstring (i)  == bitstring (i / 2) // bitstring (i mod 2);
     i >= 0 and i <= 255  ==> octet (i)   == bitstring (i) or '00000000'B;
     i < 0                ==> bitstring (i)  == raise OutOfRange;
     i < 0 or i > 255   ==> octet (i)    == raise OutOfRange;
     )))
```

By

```
/*Definition of 'xxxxx'B literals, num, bitstring and octet */
     for all s in Bitstring (
     for all b in Bit (
     for all i in Integer (
     /* definition of num */
     <<type Bitstring>>num (''B)            == 0;
     num (s // mkstring (b))                == num (b) + 2 * num (s);
     /* definition of bitstring */
     <<type Bitstring>>bitstring (0)        == '0'B;
     <<type Bitstring>>bitstring (1)        == '1'B;
     i > 1                ==> bitstring (i)  == bitstring (i / 2) // bitstring (i mod 2);
     i < 0                ==> bitstring (i)  == raise OutOfRange;
     /* definition of octet */
     i >= 128 and i <= 255  ==> octet (i)   == bitstring (i) or '00000000'B;
     i >=   0 and i <= 127  ==> octet (i)   ==
                      substring('00000000',0,8-length(bitstring(i))// bitstring(i);
     /* the bitstring is extended to the left enough zero bits to make it an octet */
     i < 0 or i > 255   ==> octet (i)    == raise OutOfRange;
     )))
```

B.9 Z.105 changes

None.

B.10 Z.106 changes

None.

B.11 Z.107changes

None.

B.12 Z.109 changes

None.

NOTE – A revision of Z.109 was sent for consent for approval at the same time as this document was sent for approval.

B.13 Z.111 changes

B.13.1 Use of graphical meta-symbols

The following should be added as an Appendix.

Use of graphical meta-symbols

The meta symbols for the notation of the graphical grammar should always be used according to their definition.

- It is required that the operators (**contains**, **is associated with**, **is followed by**, **is connected to**, **is attached to)** to have a graphical non terminal as the left hand argument, that is a non-terminal ending with the word “symbol”.
- The left-hand-side should never be empty.
- The right-hand-side can a set that may be empty, such as the form … **is connected to** {<gate>*}**set** ; that is connected to an unordered (logically and graphically), possibly empty set of <gates>.
- The post-fix operator **set** should be applied consistently. The form {<xxx>*}**set** should be used rather than {{<xxx>*}**set**} or {<xxx>}* **set** as right hand side of a graphical meta operator, where the meaning that the right hand side is an unordered set of <xxx>. In particular {<xxx>}* **set** means an ordered list of <xxx> that is in a set with just one element. In {{<xxx>*}**set**} the outer braces are (probably) redundant. The form {{<xxx>*} <yy> }**set** means an ordered list of <xxx> items with <yy> required in the set but with the placing unordered.
- If **set** is in a graphical context the items are unordered vertically or horizontally but should not overlap.
- If the right hand side of an operator is
  {<xx><yy><zz>} this is an ordered list <xx><yy><zz> and
  {<xx><yy><zz>}**set** should be used if <xx> and <yy> and <zz> are graphical
  {<xx>*} is an ordered list of <xx> and
  {<xx>*}**set** should be used if <xx> is graphical
- For a line symbol that connects two endpoints the form
  <xxx line symbol> **is connected to** {<end1>} **is connected to** {<end2>}
  should be used where possible, though the same meaning is given by
  <xxx line symbol> **is connected to** {<end1> <end2>} **set**.

B.14 Z.119 changes

None.

B.15 Open issues.

B.15.1 Additional issues to be considered

Open issues for which no contributions are available:

- Data type library extensions (predefined object types, Standard Template Library analogue)
- Memory management issues
- Instance sets vs. container types and navigation into composite agents
- Broadcast mechanisms

- Interrupts

B.15.2 List of Open Items

The following is a list of issues classified as open items according to the rules for maintenance for SDL. The list below originates November 1999 but a number of items have been subsequently removed, though at the time of approval of this document the list had not been reviewed for a considerable period and therefore should be revised.

To facilitate the tracking of open items each item is given an identifier of the form (month/year).<number> where month and year identify the meeting at which the item was first put onto the list. For example "(04/97).3".

To keep the list concise, the details given in relevant documents are not copied to the list, but the documents are referenced. However, a consequence of this procedure is that some references are to temporary documents of meetings, and therefore all referenced documents are kept as 'retained documents' on the ITU informal FTP server
(<ftp://username:password@ties.itu.int/u/tsg17/sg17/archive/2001_xchange/wp3/q13/Retained_documents/>)
and on the SDL Forum ftp server at
<ftp:// username:password@ftp.sdl-forum.org/sg17/wp3/q13/Retained_documents/>.

(04/97).13 output -, input *, reset * (TD 34, TD 37 SG 10 April 1997, note that this may also be related to signal as data);

(04/97).17 signal parameter access (TD 34, TD 35 SG 10 April 1997, e.g., refer to signal parameters in enabling conditions);

(04/97).25 more flexible USE syntax – USE p1, p2, p3 (TD 35 SG 10 April 1997);

(10/98).1 Simpler initialization of systems (TD 34 SG 10 April 1997, TDS 603 October 1998,TDN 631 November 1998) and dynamic routing;

(11/99).1 SuperType Method call (R 10 of the study period 1997-2000, SG 10, Annex 10.6, 2.2.13);

(06/00).1 Exit connection points for tasks (TDA03 6.6 June 2000)

B.15.3 List of Closed items (see B.1 for meaning of a "closed" item)

To facilitate the tracking of items each item uses the identifier of the form (month/year).<number> given when the item was first put onto the open item list. For example "(04/97).3". If the items were never on the open item list, the numbers are consecutive to the open items for the meeting at which the closed item is identified.

(10/96).9 Allow algorithmic operators with external data - requirement not clear (COM-10-1);

(10/96).13 Operators returning sets of values (multivalued operators) - can adequately be handled with STRUCT (COM-10-1);

(04/97).18 signal Priority (TD 34 SG 10 April 1997);

NOTE: In SDL-2010 it is possible to state the availability time for signals.

(04/97).30 Relaxation of the rules for signals to services, because this would break the model for services and context parameters. (TD 35 SG10 April 1997);

NOTE: services are replaced by state aggregation.

(04/97).31 **virtual** as default, because this had been extensively discussed (and rejected) when SDL-92 was formulated and has implications on the use of constraints. (TD 37 SG 10 April 1997);

(10/98).2 remote process creation, because this was added to the language, although the need can be satisfied by remote procedure and the state machine of a block and it was decided an additional construct made the language too complex (TDI 608 Internet meeting Autumn 1998);

Annex C

Master list of Changes for SDL-2000

C.1 Changes to Z.100 (11/07)

C.1.1 Defect correction - 12.3.3.1 Extended variable, Model (see COM 17 -TD 0454 (Geneva, 16-25 September 2009))

<is assigned sign> <expression> was missing.

In the Model section change

<indexed variable> is derived concrete syntax for:

to

The concrete syntax form:

<indexed variable> <is assigned sign> <expression>

is derived concrete syntax for:

This corrects the model, and makes it consistent with <field variable>.

C.2 Changes to Z.104 (10/04)

None.

C.3 Changes to Z.105 (07/03)

None.

C.4 Changes to Z.106 (08/02)

The following are changes to Z.106 (08/02):

C.4.1 Extension - A75 extended task symbol (see COM 17-TD 3202 and COM 17-TD 3225 (Geneva, 10-19 March 2004))

This change is to validate some existing CIF produced using an implementation of CIF prior to agreement of Z.106.

Add to the end of section 7.4.19 A19 CIF descriptor the alternative:

"| <extended task symbol: A75>"

Add to section 7.4.49 just before the example the note:

"NOTE <extended task symbol: A79> can be used as an alternative syntax."

After section 7.4.74 insert as a new section

"7.4.75 A75 extended task symbol:

/* CIF Extendedtask <position and size: B20> ***/**
[<text position: B21>]
<left curly bracket>
<statement list>

<right curly bracket>
<end>

Additional information:

See also <task symbol: A49> that describes an alternate syntax to represent task symbols.

Example:

/* CIF Task (800,550) */

TASK myVariable := 0;"

C.5 Changes to Z.109 (06/07)

The following are changes to Z.109(06/2007). In the following **TD0118** is COM 17-TD 0118 [2009-2012].

C.5.1 Clarification – 4 Abbreviations and Synonyms - Use of the SDL in references

The acronym SDL is used in references before paragraph numbers that refer to Z.100. This is clarified by adding after

SDL Specification and Description Language

the text

, and in particular in References clauses to the relevant Z.100 series Recommendation for the reference

C.5.2 Defect correction – 7.4 DataType, 7.4.2 Constraints – TD0118 item 1

Delete the unnecessary constraint

- interfaceRealization shall be empty.

C.5.3 Defect correction – 7.13 Signal, 7.13.2 Constraints – TD0118 item 1

Replace the unnecessary constraint by the text

No additional constraints

C.5.4 Defect correction – 8.2 PseudoState - TD0118 item 3.1

The "else" of decisions had been omitted.

In 8.2.2 Constraints after

A Transition shall have a non-empty guard property Constraint (a Boolean Expression)

add the text

and an empty trigger property

and replace

Each guard of each Transition that is an outgoing property

by

Each Boolean guard of each Transition that is an outgoing property

In 8.2.3 Semantics in the paragraph starting

A <<Pseudostate>> Pseudostate with kind == choice

replace "guard" by "Boolean guard" (twice), and add to the end of the paragraph the sentence

An outgoing property with an “else” guard property is mapped to an *Else-answer* where the *Transition* is mapped in the same way as for a Boolean guard property.

C.5.5 Defect correction – 8.5 StateMachine, 8.5.2 Constraints - TD0118 item 1

The constraint to ensure a StateMachine does not return a result for a *Composite-state-type*, was invalid. Replace the text

The returnedResult property shall be empty

by

No ownedParameter property shall have a direction=return

C.5.6 Defect correction – 8.6 Transition, 8.6.2 Constraints - TD0118 item 3.1

An empty trigger property is valid, so delete the constraint

The trigger property shall not be empty.

C.5.7 Clarification – 8.6 Transition, 8.6.3 Semantics - TD0118 item 3.1

To make it clear how a Transition is handled with it is the outgoing property of a <<Pseudostate>> Pseudostate with kind == choice, add a new paragraph to the start of 8.6.3 Semantics

A <<Transition>> Transition that is the outgoing property of a <<Pseudostate>> Pseudostate with kind == choice is mapped as defined for Pseudostate in 8.2.3.

and after

If a <<Transition>> Transition has a non-empty trigger property and non-empty guard property

add

and is not the outgoing property of a <<Pseudostate>> Pseudostate with kind==choice

C.5.8 Defect Correction – 9.11 Decision - TD0118 item 3.2

To handle the "else' clause of a decision, in 9.11.2 Constraints add the constraints and note

- For every Clause except the “else” Clause, the predecessorClause set shall be empty, so that there is no requirement that any Clause is evaluated before any other Clause (except the “else” Clause). The predecessorClause set for an “else” Clause shall include every other Clause, so that they all have to be evaluated before the “else” Clause. For every Clause except the “else” Clause, the successorClause set shall contain only the “else” Clause if there is one, otherwise the successorClause set shall be empty, because the order of evaluation is never enforced in SDL-2010. The successorClause set of the “else” Clause shall be empty.

NOTE – The “else” Clause is a Clause that is a successor to all others and whose test part always returns true, so that it is only invoked if all others are false (see UML-SS 12.3.18 ConditionalNode).

- The isAssured property shall be true. Therefore either there is shall be an “else” Clause, or for every possible value of the left hand side of the expressions (the *Decision-question*) there shall be at least one test that succeeds.

Also in the second sentence of 9.11.3 Semantics, after "The Clause" add

set (excluding the “else” Clause)

and at the end of 9.11.3 Semantics add the sentence

The “else” Clause (if present) defines the *Else-answer*, otherwise there is no *Else-answer*.

Because an asterisk is not allowed for an else part, in 9.11.5 Notation after the syntax for <algorithm answer part>, add the paragraph

A <range condition> in an <algorithm answer part> shall not contain an <asterisk>.

and in 9.11.5 Notation after the syntax for <question>, insert the note, paragraph and references section that had been wrongly placed at the end of 9.20.5 (delete them from 9.20.5)

NOTE – The syntax for <range condition> is given in clause 7.12.4, the notation for Property.

The <question> represents the left-hand side of every test expression (the question). The <range condition> of each <algorithm answer part> determines the operator for the expression and the value for the right-hand side of the expression. If the <range condition> consists of a single <open range>, the operator is the operator corresponding to the <equality sign>, <not equals sign>, <less than sign>, <greater than sign>, <less than or equals sign>, or <greater than or equals sign>. Otherwise, <range condition> is evaluated to a set value that contains the values specified by the <range condition> and the operator is the membership operator for the left-hand side of the test being in this set. The type of the set is the set of all possible values of the type of the left-hand side of the test. The <statement> of the <algorithm answer part> represents the body of the Clause with the test. The test for the <algorithm else part> is the question not being in the set of values covered by any of the right-hand sides (that is the test is true only if all other tests are false). The <statement> of the <algorithm else part> represents the body of the else Clause.

9.11.5 References

SDL: 11.13.5 Decision

11.14.5 Decision statement

C.5.9 Defect Correction – 9.14 If, 9.14.2 Constraints - TD0118 item 3.3

Z.109 does not use OpaqueExpression, so replace "OpaqueExpression" by "<<Expression>> Expression" (twice).

C.5.10 Defect Correction – 9.15 LoopNode, 9.14.2 Constraints - TD0118 item 3.3

Z.109 does not use OpaqueExpression, so replace "OpaqueExpression" by "<<Expression>> Expression" (once).

C.5.11 Clarification – 9.16 OpaqueAction - TD0118 item 2

Change the first NOTE to NOTE 1, and add a NOTE 2

NOTE 2 – The body elements (if any) of an OpaqueAction are not used to derive the SDL-UML semantics of the OpaqueAction, instead the semantics depend on the actual OpaqueAction subtype and its attributes. However, if one element of the language of the OpaqueAction is [ITU-T Z.100], the corresponding body should be the string for the notation of the OpaqueAction defined in textual syntax.

C.5.12 Clarification – 9.20 SendSignalAction- TD0118 item 3.2

Delete the text wrongly placed after the 9.20.5 References, which is moved to 9.11 Decision (see above).

_____