INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.100
Addendum 1
(10/96)

SERIES Z: PROGRAMMING LANGUAGES

Specification and Description Language (SDL)

CCITT Specification and Description Language (SDL)

Addendum 1

ITU-T Recommendation Z.100 – Addendum 1

(Previously CCITT Recommendation)

ITU-T Z-SERIES RECOMMENDATIONS

PROGRAMMING LANGUAGES

Specification and Description Language (SDL)	**Z.100–Z.109**
Criteria for the use and applicability of formal Description Techniques	Z.110–Z.199
ITU-T High Level Language (CHILL)	Z.200–Z.299
MAN-MACHINE LANGUAGE	Z.300–Z.499
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.399
Miscellaneous	Z.400–Z.499

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

FOREWORD

The ITU-T (Telecommunication Standardization Sector) is a permanent organ of the International Telecommunication Union (ITU). The 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 Conference (WTSC), which meets every four years, establishes the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 1 (Helsinki, March 1-12, 1993).

ITU-T Recommendation Z.100, Addendum 1, was prepared by ITU-T Study Group 10 (1993-1996) and was approved by the WTSC (Geneva, 9-18 October 1996).

NOTE

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

CONTENTS

SUMMARY

This is an addendum to Recommendation Z.100 (1993). They both describe the 1996 version of the CCITT Specification and Description Language (SDL). This Addendum contains corrections to previous language definition and some additions and changes to the language. There will be no additional Formal Definition of these changes.

BACKGROUND

During the Study Period from 1993 to 1996, a number of minor errors were identified and added to a living "Master List of Changes". The Study Period was accompanied by a strong request for stability of the language SDL and this was respected. However, some changes and additions have been requested by users. These have been taken into account. The publishing of only an Addendum instead of a new Recommendation underlines that stability is implied in this revised language version.

Recommendation Z.100

CCITT SPECIFICATION AND DESCRIPTION LANGUAGE (SDL)

ADDENDUM 1

(Geneva, 1996)

1 Scope

This Addendum defines the CCITT's Specification and Description Language (SDL).

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; all 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.

[1] ITU-T Recommendation Z.100 (1993), *CCITT Specification and Description Language (SDL).*

3 Abbreviations

This Addendum uses the following abbreviation.

SDL Specification and Description Language

4 General remarks on the list of corrections to Recommendation Z.100

This Addendum reflects the master list of changes that have been accumulated during the Study Period from 1993 to 1996 and includes all those changes that have been approved by ITU-T Study Group (SG) 10 at its meetings. The master list of changes had entries in the following categories:

- *Corrections* and *Clarifications*: considered to take effect immediately and were described in full;
- *Modifications, Decommitted features* and *Extensions*: agreed by SG 10 as candidates to be included in a revision of Recommendation Z.100, but subject to the full ITU approval procedures and only take effect when new or revised Recommendations are approved by the ITU;
- *Open items* and *Closed items*: record the current status of study issues.

There are some *Modifications*, *Extensions* and *Open items* that require further study.

The SDL maintenance terminology, guidelines and change request procedure can be found in clause 15 of this Addendum.

5 Corrections and clarifications to Recommendation Z.100

Subclause 2.2.2

Change the production rule for <path item> on page 17 for the concrete textual grammar adding the option of "<operator name> <exclamation>":

```
<path item> ::=
                    <scope unit kind>
                    { <name> | <quoted operator> | <operator name> <exclamation> }
```

Subclause 2.2.2

Add in *Semantics* at the end of the subclause before the last two paragraphs:

The following clarifies the visibility rules for procedures, services and service types (at the end of p. 19):

1) Procedures, services and service types defined locally to a process can freely access and use the variables and signals of that process.

2) Globally defined procedures, services and service types must not use any variables defined in the calling context.

3) To access and use entities defined locally to the calling process, corresponding context parameters must be used.

Subclause 2.4.5

Modify the last line of the penultimate paragraph of the *Concrete textual grammar* (p. 38, 2nd line) to:

An exported procedure defined in the enclosing process must be mentioned by an input or save in exactly one service.

Subclause 2.4.5

In the service text area, <valid input signal set> has been omitted. This is included in the textual representation and in the corresponding graphical representation for process diagram (<process text area>). The corrected production rule is:

```
<service text area> ::=
        <text symbol> ***contains*** {
          [ <valid input signal set> ]
          { <variable definition>
                  ....
          | <macro definition>  } *
                              }
```

Subclause 2.4.6

In the procedure body, the textual representation has an error, whereas the graphical representation has the intuitively correct production. The correct production should be:

```
<procedure body> ::=
        [ <start> ]   { <state> | <free action> } *
```

Subclause 2.7.5

The first paragraph after the productions for the *concrete textual grammar* on page 68 for *Decision* should be changed to (changes underlined):

An <answer part> or <else part> in a decision or an option is a terminating <answer part> or <else part> respectively if it contains a <transition> where a <terminator statement> is specified, or contains a <transition string> whose last <action statement> contains a terminating decision or option. A <decision> or <transition option> is a terminating decision and terminating option respectively, if each <answer part> and <else part> in its <decision body> is a terminating <answer part> or<else part> respectively.

The first paragraph of the *Model* on page 69 for *Decision* should be changed to:

If a <decision> is not terminating then it is derived syntax for a <decision> wherein all not terminating <answer part>s and the <else part> if not terminating have inserted at the end of their <transition> a <join> to the first <action statement> following the decision or if the decision is the last <action statement> in a <transition string> to the following <terminator statement>.

Subclause 4.10

The model of priority input is modified. The sixth sentence of the second paragraph on page 105 is changed as follows (changes underlined):

> Priority inputs to the original state are connected to the first state, while all inputs, including priority inputs are connected to the second state (State 2). All inputs other than priority inputs are saved in the first state.

Subclause 4.14

The non-terminal <end> is removed in the corrected production (p. 113):

<remote procedure save area> ::=
<save symbol> ***contains***
{[<virtuality>] **procedure** <remote procedure identifier list> }

Subclause 5.3.1.11

(p.139) in the *Example* of *Model* the axioms part should be (delete a ")"):

axioms
Exor (a,b) == (a and (not b)) or ((not a) and b);

Subclause 5.3.2

(p.147) the non-terminal <sort> replaces extended sort in the following:

<operator result> ::=
returns [<variable name>] <sort>

Subclause 5.4.6

The rule for <external properties> should be

<external properties> ::=
alternative
<external formalism name> [, <word>] <end>
<external data description>
[endalternative] <end>

Subclause 6.1.2

(type expression) in the *Semantics* list of cases, the words "or view definition" of the fourth item should be deleted as it does not apply.

Subclause 6.2

Change the penultimate paragraph of the *Concrete textual grammar* (p.178) to:

> The binding of an actual variable or synonym context parameter to its definition is not resolved by context but by using the visibility rules.

Subclause 6.3.2

Recommendation Z.100 defines where virtual types are allowed inside a virtual block, and the implicit constraint for every case. Some unclear situations have been pointed out. To clarify this issue, the following two sentences are proposed as additions to 6.3.2.

Addition to the static conditions of the *concrete textual grammar*

> If both **inherits** and **atleast** are used then the inherited type must be identical to or be a subtype of the constraint.
>
> In the case of an implicit constraint, redefinition involving **inherits** must be a subtype of the constraint.

6 Minor extensions to Recommendation Z.100

Subclause 2.2.2

Add to the end of the paragraph that begins with "It is allowed to omit ...":

> With respect to visibility, if the <identifier> does not contain a <qualifier>, a <package reference clause> is considered as the nearest enclosing scope unit to the scope unit to which it is associated and contains the entities visible from the package.

Subclause 2.2.6

An additional comment symbol will be added as an option requiring a change in the *Concrete Graphical Grammar:*

<comment area> ::=

{<comment symbol> | <comment symbol2>} ***contains*** <text>

is connected to <dashed association symbol>

Further, add after the definition of the <comment symbol>:

<comment symbol2> ::=

7 Harmonization of signals and remote entities

7.1 Motivation

Remote procedure definitions and remote variable definitions serve the same purpose as signal definitions: To define communication primitives. However, compared to signals, there are in SDL-92 two limitations on the use of remote procedures and remote variables;

1) They cannot be mentioned on the interfaces (signal routes, channels and gates).

2) They cannot be communicated with the environment of the system.

In SDL-96 remote procedures and remote variables are harmonized with signals. This gives increased expressive power of SDL and it provides a more coherent approach for specification of interfaces (and thereby makes SDL less complex).

Ability to communicate with the environment through use of remote procedure calls is needed for two reasons:

1) When building large systems it must be possible to regard a part of the system as a system of its own in order to apply testing, reuse and encapsulation principles properly. Such a part will typically be a block substructure, as it resembles a whole system, but it will be a major problem to regard a block substructure as a sub-system, because block substructures, as opposed to systems, are allowed to communicate with their environment using remote procedures and variables.

2) When using SDL for distributed computing architectures such as the one defined by the TINA consortium, the number of servers (which may be SDL systems) offering interfaces to clients (which also may be SDL systems) may change during execution. For SDL-92, usage in this area is difficult.

7.2 Solution

Remote procedures and remote variables can be specified in signal lists, such that they, like signals, can be specified in channels, signal routes, gates, signalset and save. Those scope units which have remote procedures and remote variables on the connected gates, channels or signal routes need not include <imported procedure specification>s and <imported variable specification>s. To communicate remote entities with the environment, they must be mentioned on the channels connected to the system environment.

Also, <basic input part> is harmonized with <basic save part> as a <basic input part> in SDL-96 consists of a list of <signal list item>s. In other words, a signal list identifier may be mentioned in an input in SDL-96. The two extensions together eliminate the need for the productions <remote procedure save> and <remote procedure input>.

If an identifier in a <signal list item> both can denote a signal and a remote procedure/variable according to the visibility rules, the identifier denotes the signal, unless the identifier is preceded by the keyword **procedure** (for a remote procedure) or **remote** (for a remote variable).

7.3 Example

signal s1, s2(integer);

remote procedure rp;

signallist slist = s1, s2, **procedure** rp; /* The keyword **procedure** is not needed if no visible signal is named rp */

...

gate g **in with** (slist);

...

input (slist);

...

7.4 Changes

Subclause 1.3.2:

First line: change "received from" to "exchanged with"

Third line: change "signals" to "stimuli"

Subclause 2.1

Line 2: change "by signals" to "by means of signals and remote procedures/variables"

Subclause 2.4.1.2

In production <entity kind> change

| procedure

into

| [remote] procedure

Subclause 2.4.1.2

Add after production for <interface>:

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

Subclause 2.4.2

Third line in *Semantics*: Replace "only take place using signals. Within a system, these signals are conveyed on channels" by "take place using signals, remote procedures and remote variables. Within a system, these entities are conveyed on explicit or implicit channels"

Subclause 2.4.4

Page 36, line 5 from top: replace "any signal" by "any stimulus"

Page 36, Add after paragraph starting with "Signals received by process instances ...":

> "Calling and serving remote procedure calls and accessing remote variables also correspond to exchange of signals, see 4.13 and 4.14"

Subclause 2.5.1

First line of *Semantics:* replace "signals" by "signals (including the implicit signals implied from exchange of remote procedures and remote variables, see 4.13 and 4.14)."

Subclause 2.5.2

First line of *Semantics:* replace "signals" by "signals (including the implicit signals implied from exchange of remote procedures and remote variables, see 4.13 and 4.14)."

Subclause 2.5.2

Add a paragraph in *Semantics:*

> Remote procedures and remote variables need not be mentioned in the signal lists of the signal routes and channels between the importer and exporter, but if a remote variable is mentioned in a channel or signal route, then it must be mentioned in the whole communication path between the importer and exporter. An imported procedure or variable is mentioned as outgoing from the importer and an exported procedure or variable is mentioned as incoming to the exporter. If an imported procedure or variable is not mentioned in the signal route outgoing from a process or service or mentioned in the gates of a process type or service type, then it must be specified in an <imported procedure specification> or <imported variable specification> in that process (type) or service (type).

Page 47, 3rd line in 2nd paragraph change:

> "<output>s and" by "<output>s, <exported variable definition>s, <procedure preamble>s, <imported procedure specification>s, <imported variable specification>s and"

Page 47, 3rd line in 3rd paragraph change:

> "<output>s and" by "<output>s, <exported variable definition>s, <procedure preamble>, <imported procedure specification>s, <imported variable specification>s and"

Page 47, 2nd line in 5th paragraph change:

> "output>s and" by "<output>s, <exported variable definition>s, <procedure preamble>s, <imported procedure specification>s, <imported variable specification>s and"

Page 47, 3rd line in 6th paragraph change:

> "<output>s and" by "<output>s, <exported variable definition>s, <procedure preamble>s, <imported procedure specification>s, <imported variable specification>s and"

Page 47, 3rd line in 7th paragraph change:

> "an output" by "the outputs, imported procedure specifications and imported variable specifications"

Page 47, 3rd line in 8th paragraph change:

"and the <output>s and" by "the <output>s, the <exported variable definition>s, <procedure preamble>s, <imported procedure specification>s and <imported variable specification>s"

Subclause 2.5.5

Change "signal identifiers" by "signal identifiers, remote procedures, remote variables"

Subclause 2.5.5

Change <signal list item> and the following paragraph in *Concrete textual Grammar:*

<signal list item> ::=

<signal identifier>

| (<signal list identifier>)

| <timer identifier>

| [**procedure**] <remote procedure identifier>

| [**remote**] <remote variable identifier>

The <signal list> which is constructed by replacing all <signal list identifier>s in the list by the list of <signal identifier>s or <timer identifier>s they denote and by replacing all the <remote procedure identifier>s and all the <remote variable identifier>s by one of the implicit signals each of them denotes (see 4.13 and 4.14), corresponds to a *Signal-identifier*-**set** in the *Abstract grammar*.

A <signal list item> which is an <identifier> denotes a <signal identifier> or <timer identifier> if this is possible according the visibility rules and else a <remote procedure identifier> if this is possible according to the visibility rules and else a <remote variable identifier>. To force a <signal list item> to denote a <remote procedure identifier> or <remote variable identifier> the keyword **procedure** or **remote** respectively can be used.

Subclause 2.6.4

Concrete textual grammar delete <input part> and rename <basic input part> to <input part>.

Change *Concrete textual grammar* for <stimulus> to

<stimulus> ::=

<signal list item> [([<variable>] {,[<variable>]}*)]

A <signal list item> must not denote a <remote variable identifier> and if it denotes a <remote procedure identifier> or a <signal list identifier>, the <stimulus> parameters (including the parenthesis) must be omitted.

Subclause 2.6.4

Add to *Model:*

A <stimulus> whose <signal list item> is a <signal list identifier>, is derived syntax for a list of <stimulus>s without parameters and inserted in the enclosing <input list> or <priority input list>). In this list there is a one-to-one correspondence between the <stimulus>s and the members of the signal list.

Subclause 2.6.4

Add after 1st paragraph in *Concrete textual grammar:*

In the *Abstract Grammar*, the <remote procedure identifier>s are also represented as *Signal-identifier*s.

Subclause 2.6.4

Delete <input part> and rename <basic input part> to <input part>.

Subclause 2.6.4

Delete <input area> and rename <basic input area> to <input area>.

Subclause 2.6.5

1st paragraph: change "signal identifiers" to "signal identifiers and remote procedure identifiers".

Subclause 2.6.5

Delete <save part> and rename <basic save part> to <save part>.

Subclause 2.6.5

Delete <save area> and rename <basic save area> to <save area>.

Subclause 4.2.2

Delete <remote procedure input area> and <remote procedure save area> as alternatives in <any area>.

Subclause 4.13

Change <import expression> into:

<import expression> ::=

import (<remote variable identifier> [,<destination>] [**via** <via path>])

Subclause 4.13

Model 1st line: change "conveyed on implicit channels and signal routes." to "conveyed on implicit or explicit channels and signal routes. The channels and signal routes are explicit if the remote variable has been mentioned in the <signal list> (outgoing for the importer and incoming for the exporter) of at least one gate, channel or signal route connected to the importer or exporter. When a remote variable is conveyed on explicit channels and signal routes, the **nodelay** keyword from the <remote variable definition> is ignored.

Subclause 4.14

Concrete textual grammar: delete <remote procedure input transition> and <remote procedure save>

Subclause 4.14

Change <remote procedure call body> into:

<remote procedure call body> ::=

<remote procedure identifier> [<actual parameters>]

[**to** <destination>] [**via** <via path>]

Subclause 4.14

Concrete textual grammar add:

When the <destination> and the <via path> are omitted, there is syntactic ambiguity between <remote procedure call body> and <procedure call>. In this case, the contained <identifier> denotes a <procedure identifier> if this is possible according to the visibility rules and otherwise a <remote procedure identifier>.

Subclause 4.14

Concrete graphical grammar: delete <remote procedure input area> and <remote procedure save area>

Subclause 4.14

last paragraph before *Model:* change two occurrences of <remote procedure save> into <save> and two occurrences of <remote procedure input transition> into <input part>.

Subclause 4.14

Model: 1st line change "conveyed on implicit channels and signal routes." to "conveyed on implicit or explicit channels and signal routes." The channels and signal routes are explicit if the remote procedure has been mentioned in the <signal list> (the outgoing for the importer and the incoming for the exporter) of at least one gate, channel or signal route connected to the importer or exporter. When a remote procedure is conveyed on explicit channels and signal routes, the **nodelay** keyword from the <remote procedure definition> is ignored.

Subclause 4.14

Page 114, item a): change two occurrences of "**to** destination" by "**to** destination **via** viapath"

Subclause 4.14

Page 114: delete paragraph starting with "A <remote procedure input transition> or <remote procedure input area> ...

Delete **remote procedure input** and **remote procedure save** from the glossary.

8 External procedures and external operators

8.1 Motivation

SDL is both used as a specification language and as an implementation language. However, there are situations where SDL should not or cannot be used. Such situations include:

- when it is convenient to reuse some existing code, written in another language;
- when interfacing to an external program (e.g. the Windows API), and the external program does not define a language binding to SDL (typically, an external program demands C++ or Pascal to be used, if an external interface is supported at all);
- when low-level code is needed, such as in development of device drivers; or
- when working with pointers.

By allowing calls in SDL to procedures and operators which are developed using another language than SDL, it is possible to use SDL as implementation language more generally.

8.2 Solution

It is proposed to introduce external procedures and operators in SDL. Like for external synonyms and external data, the semantics of calling an external procedure will not be in the scope of SDL.

When SDL is used as a specification language, external procedures and operators can be used for expressing that the effect of a given call is not within the scope of the specification. When SDL is used as an implementation language, external procedures and operators can be used for interfacing to other languages.

The formal of an external procedure/operator should be the same as the format of procedure/operator reference, but with the keyword **referenced** replaced by **external**. No special graphical syntax should be used when defining external procedures (as similar to remote procedures), but in order to allow full type checking, the formal parameters and result type must be given in the definition (this is already the case for operator references in SDL-92).

8.3 Example

SDL has no built-in features for ordering of character strings, so a C library function could be added somewhere:

procedure strcmp **fpar** op1, op2 Characterstring; **returns** Integer; **external**;

8.4 Changes

Changes to **Subclause 2.4.6**:

Add <external procedure definition> as alternative in <procedure definition>, i.e.:

<procedure definition> ::=

<external procedure definition> |

<procedure preamble>

procedure {<procedure name> | <procedure identifier> }

...

endprocedure [<procedure name> | <procedure identifier>] <end>

Add to *Concrete Textual Grammar:*

<external procedure definition> ::=

procedure <procedure name>

[<procedure signature>] **external** <end>

An external procedure cannot be mentioned in a <type expression>, in a <formal context parameter> or in an <atleast constraint>.

There is no corresponding graphical syntax for <external procedure definition>.

Add to *Semantics:*

An external procedure is a procedure whose <procedure body> is not included in the SDL description. Conceptually, an external procedure is assumed to be given a <procedure body> and to be transformed into a <procedure definition> as part of the generic system transformation, see Subclause 4.3.1.

Subclause 5.3.2

Add <external operator definition> as alternative in <operator definition>, i.e.:

<operator definition> ::=

<external operator definition> |

operator {<operator identifier> | <operator name> }

...

endoperator [<operator identifier> | <operator name>] <end>

Add to *Concrete Textual Grammar:*

<external operator definition> ::=

operator <operator name>

[<operator signature>] **external** <end>

There is no corresponding graphical syntax for <external operator definition>.

Add to *Semantics:*

An external operator is an operator whose behaviour is not included in the SDL description. Conceptually, an external operator is assumed to be given behaviour and to be transformed into an <operator definition> as part of the generic system transformation, see subclause 4.3.1.

9 Extensions and harmonization on channels and signal routes

The extensions and harmonization issues on channels and signal routes are:

1) Allowing channels to be optional.

2) Simplifying the specification of channels and signal routes.

3) Channels/signal routes connected to same scope unit in both endpoints.

9.1 Allowing channels to be optional

Like for signal routes, it is in SDL-96 optional to specify the channels for a given scope unit.

The reasons for this extension are:

- In systems consisting of many blocks, the channels reduces readability and it makes it difficult to split a system/substructure over several pages.
- For open client/server bases systems, focus is more on the interfaces offered/consumed by the server/client rather than the (dynamically changing) communication links.

9.2 Simplifying the specification of channels and signal routes

Also when channels/signal routes are specified for a given scope unit, it is allowed to leave out some of its 'redundant' information, thereby simplifying the diagrams:

- *The channel/signal route name*
  Often, the channel or signal route name is not referred anywhere. In particular, this is the case when connecting to type-based instances because in such cases there is no channel-to-signalroute connections and the gate identifier is used in **output via** instead of the channel/signal route.
- *The signal lists*
  Often the signal lists of a channel/signal route can be derived from its endpoints. In particular this is the case, when connecting type-based instances and when making 1-to-1 connections of instances.

9.3 Channels/signal routes connected to same scope unit in both endpoints

To allow a channel or a signal route to be connecting to the same scope unit in both endpoints is an already recognized requirement to SDL-96 as it is listed as an open issue for SDL-92.

In SDL-96, uni-directional (not bi-directional) channels and signal routes can be connected to the same scope unit in both endpoints (which must not be **env**).

The reasons why such channels and signal routes must be uni-directional are to overcome ambiguity problems related to block sets, channel substructures and connections to type-less instances.

9.4 Changes

These changes allow the following:

- To connect both endpoints of a channel or signal route to the same block, process set or service.
- To omit channels from a scope unit.
- To omit the name of a channel or signal route from the definition of the channel or signal route.
- To omit one or both signal lists from a channel or signal route.
- To mention the same channel or signal-route name at several places at the boundary of a block, process or service diagram.

Subclause 2.4.4

Concrete textual grammar: The penultimate paragraph is replaced by:

> "The use of <valid input signal set> is defined in 2.5.2, *Concrete textual grammar*, and in 7.3."

Subclause 2.4.5

Concrete textual grammar: The last paragraph is replaced by:

> "The complete <valid input signal set>s (see 7.3.1) of the services within a process must be disjoint. Furthermore, no two services within a process may consume instances of the same timer type."

Subclause 2.5.1

Abstract grammar: In the 1st paragraph after the grammar rules, the last sentence is replaced by:

> "If both endpoints are the same block, the channel must be unidirectional (i.e. the second *Channel-path* in *Channel-definition* must be absent)."

Subclause 2.5.1

Concrete textual grammar: In the grammar rule for <channel definition>, "<channel name>" is replaced by "[<channel name>]".

The grammar rule for <channel path> is changed to:

> <channel path> ::=
>
> **from** <channel endpoint> to <channel endpoint>
>
> [**with** <signal list>] <end>

The following paragraph is added immediately after the grammar rules:

> "The ending <channel name> may only be specified if the starting <channel name> is specified. If the starting <channel name> is not specified, the channel cannot be referred to by name."

The following paragraph is added immediately before *Concrete graphical grammar*:

> "A channel is allowed to connect the two directions of a bi-directional gate to each other."

Subclause 2.5.1

Concrete graphical grammar: In the grammar rule for <channel definition area>, "<channel name>" is replaced by "[<channel name>]", and the first occurrence of "<signal list area>" is replaced by "[<signal list area>]".

In the 1st paragraph after the grammar rules, the first occurrence of "a <signal list area>" is replaced by "at most one <signal list area>".

Subclause 2.5.1

Semantics: In the 1st paragraph, the first sentence is replaced by:

> "A *Channel-definition* represents a transportation route for signals (including the implicit signals implied from exchange of remote procedures and remote variables, see 4.13 and 4.14)."

The last paragraph is deleted.

Subclause 2.5.1

Model: The whole text is replaced by :

> "*Implicit channel names*: If the <channel name> is omitted from a <channel definition> or <channel definition area>, the channel is implicitly and uniquely named unless the channel has a substructure associated. In this case, the <channel name> is the same as the <channel substructure name>.

Implicit channels and implicit signal lists on channels: If a system, system type, block substructure or channel substructure contains no explicit channels, implicit channels without signal lists are derived. Thereafter, channels without explicit signal lists are filled in with signals, remote procedures and remote variables. The details of this are described in 7.3.3 and 7.3.5.

If a system, system type, block substructure or channel substructure contains explicit channels for signals, but no explicit channels for remote procedures and remote variables, implicit channels are derived which are able to carry the remote procedures and remote variables in question. The details of this are described in 7.3.4.

Explicitly defined <channel path>s must have non-empty signal lists after these transformations.

Channels connected to type-based block sets: A channel with both endpoints being gates of one <textual typebased block definition> represents individual channels from each of the blocks in this set to all blocks in the set, including the block itself. Any resulting bi-directional channel connecting a block in the set to the block itself is split into two unidirectional channels.

A channel with one endpoint being a gate of a <textual typebased block definition> represents individual channels to or from each of the blocks in the set.

The individual channels for each original channel all have the same delaying property as the original channel."

Subclause 2.5.2

Abstract grammar: The 2nd paragraph after the grammar rules is replaced by:

"If both endpoints are the same process set or service, the signal route must be unidirectional (i.e. the second *Signal-route-path* in *Signal-route-definition* must be absent)."

The following paragraph is added immediately before *Concrete graphical grammar*:

"A signal route is allowed to connect the two directions of a bi-directional gate to each other."

Subclause 2.5.2

Concrete textual grammar: In the grammar rule for <signal route definition>, "<signal route name>" is replaced by "[<signal route name>]".

The grammar rule for <signal route path> is changed to:

<signal route path> ::=

from <signal route endpoint> **to** <signal route endpoint>

[**with** <signal list>] <end>

The following paragraph is added immediately after the grammar rules:

"If the <signal route name> is not specified, the signal route cannot be referred to by name."

The following text is added immediately before *Semantics:*

"The following rules apply if the <block definition> or <block type definition> directly enclosing a <process definition> contains no <signal route definition>s, or if some of the <signal route path>s leading to the <process definition> do not contain <signal lists>s:

- If the <process definition> does not contain services, it must contain an explicit <valid input signal set>.
- If the <process definition> contains services, each <service definition> must either contain an explicit <valid input signal set>, or all <signal route path>s leading to the <service definition> must contain explicit <signal list>s.

A <service definition> must contain an explicit <valid input signal set> if the enclosing <process type definition> or <process definition> contains no <signal route definition>s, or if some of the <signal route path>s leading to the <service definition> do not contain <signal list>s.

If the <valid input signal set> of a <process type definition>, <process definition>, <service type definition> or <service definition> is specified explicitly, the following rules apply:

- The <valid input signal set> must be a subset of the complete <valid input signal set> (see 7.3.1).

- In case of a process type or service type, the <valid input signal set> need not mention signals which are explicitly mentioned on incoming gates of the process type or service type. In case of a process set or service, the <valid input signal set> need not mention signals which are explicitly mentioned on signal routes leading to the process set or service.

- The <valid input signal set> need not mention remote procedures or remote variables.

If a remote procedure or remote variable is not mentioned in any signal routes outgoing from a process set or service, or in any gate outgoing from a process type or service type, then it must be mentioned in an <imported procedure specification> or <imported variable specification> in that process set, process type, service or service type."

Subclause 2.5.2

Concrete graphical grammar: In the grammar rule for <signal route definition area>, "<signal route name>" is replaced by "[<signal route name>]", and the first occurrence of "<signal list area>" is replaced by "[<signal list area>]".

In the 2nd paragraph after the grammar rules, the first occurrence of "a <signal list area>" is replaced by "at most one <signal list area>".

Subclause 2.5.2

Semantics: In the 1st paragraph, the first sentence is replaced by:

"A *Signal-route-definition* represents a transportation route for signals (including the implicit signals implied from exchange of remote procedures and remote variables, see 4.13 and 4.14)."

The penultimate paragraph is replaced by:

"When a signal instance is sent to an instance of the same process instance set, interpretation of the *Output-node* either implies that the signal is put directly in the input port of the destination process, or that the signal is sent via a signal route which connects the process instance set to itself."

The following paragraph is added:

"A remote procedure or remote variable on a signal route is mentioned as outgoing from an importer and incoming to an exporter."

Subclause 2.5.2

Model: The whole text is replaced by:

"*Implicit signal-route names*: If the <signal route name> is omitted from a <signal route definition> or <signal route definition area>, the signal route is implicitly and uniquely named.

Implicit signal routes and implicit signal lists on signal routes: If a block, block type, process set (containing services) or process type (containing services) contains no explicit signal routes, implicit signal routes without signal lists are derived. Thereafter, signal routes without explicit signal lists are filled in with signals, remote procedures and remote variables. The details of this are described in 7.3.3 and 7.3.5.

If a block, block type, process set (containing services) or process type (containing services) contains explicit signal routes for signals, but no explicit signal routes for remote procedures and remote variables, implicit signal routes are derived which are able to carry the remote procedures and remote variables in question. The details of this are described in 7.3.4.

Explicitly defined <signal route path>s must have non-empty signal lists after these transformations."

Subclause 2.5.3

Concrete textual grammar: The following text is added immediately after the grammar rules:

"No channel or signal route may be mentioned more than once in a <channel to route connection> or <signal route to route connection>. No signal route may be mentioned after the keyword **and** in more than one <channel to route connection> or <signal route to route connection> of a given scope unit.

Two <channel to route connection>s or <signal route to route connection>s of a given scope unit must – before the keyword **and** – either mention the same set of channels/signal routes, or have no channels/signal routes in common."

Subclause 2.5.3

Concrete graphical grammar: The last paragraph is deleted.

The following paragraph is added at the end of the subclause:

"NOTE – Because of the **connect** construct, a channel or (external) signal route which can be anonymous in the graphical version of a specification, may need to have a name in the corresponding textual version. This is completely analogous to the case of <merge area>s in process, service or procedure graphs.

A tool which converts the graphical version of a specification to a textual one should thus be able to generate implicit channel and signal route names."

Subclause 2.5.3

A new subclause called *Model* is added and contains the following text:

"If more than one <channel to route connection> or <signal route to route connection> of a scope unit mention the same channels/signal routes before the keyword **and**, these connections are merged."

Subclause 3.2.2

Concrete textual grammar: The penultimate paragraph is replaced by:

"If a <block substructure definition> contains <channel definitions> (all with explicit <signal list>s) and <textual typebased block definition>s, each gate (only containing signals) of the <block type definition>s of the <textual typebased block definition>s must be connected to at least one channel.

If a <block substructure definition> contains <channel definitions> (some or all with implicit <signal list>s or mentioning remote procedures or remote variables) and <textual typebased block definition>s, each gate of the <block type definition>s of the <textual typebased block definition>s must be connected to at least one channel.

<channel connection>s must fulfil similar static conditions as <channel to route connection>s and <signal route to route connection>s (see 2.5.3, *Concrete textual grammar*)."

The following text is added immediately before *Concrete graphical grammar:*

> "An explicit <channel path> without an explicit signal list may have an endpoint at a block which has both an unpartitioned and a partitioned version. In this case, and after implicit <channel path>s and <signal route path>s without <signal list>s have been derived (7.3.3 and 7.3.4), the <signal list> in the <channel path> can be filled in (7.3.5) in two different ways:
>
> - As part of a sequence (of <channel path>s and <signal route path>s without explicit <signal list>s) leading into or out from the *unpartitioned* version.
> - As part of a sequence (of <channel path>s and ...) leading into or out from the *partitioned* version.
>
> The resulting <signal list> must contain the same signals, remote procedures and remote variables in both cases."

Subclause 3.2.2

Model: The following paragraph is added immediately after the existing one:

> "If more than one <channel connection> of a block substructure mention the same channels before the keyword **and**, these <channel connection>s are merged."

Subclause 3.2.3

Concrete textual grammar: The paragraph immediately after the grammar rule for <channel substructure definition> is replaced by:

> "The <channel substructure name> after the keyword **substructure** may only be omitted if the enclosing <channel definition> has an explicit <channel name>. In this case, the <channel substructure name> is the same as the <channel name>."

In the penultimate paragraph: the text ", and for each endpoint there must be exactly one <channel endpoint connection>" is deleted.

The following text is inserted between the two last paragraphs:

> "A channel may contain a <channel path> without an explicit signal list and have a channel substructure associated. In this case, and after implicit <channel path>s and <signal route path>s without <signal list>s have been derived (7.3.3 and 7.3.4), the <signal list> in the <channel path> can be filled in (7.3.5) in three different ways:
>
> - As part of a sequence (of <channel path>s and <signal route path>s without explicit <signal list>s) "bypassing" the channel substructure.
> - As part of a sequence (of <channel path>s and ...) leading into the channel substructure.
> - As part of a sequence (of <channel path>s and ...) leading out from the channel substructure.
>
> The resulting <signal list> must be the same in all three cases."

Subclause 3.2.3

Concrete graphical grammar: In the grammar rule for <channel substructure diagram>, "+" is replaced by "*".

The paragraph immediately after this grammar rule is replaced by:

> "An occurrence of <block identifier> or **env** identifies an endpoint of the partitioned channel. The occurrence is placed outside the <frame symbol> close to the endpoint of some or all of the associated subchannels at the <frame symbol>. A <channel symbol> within the <frame symbol> and connected to it indicates a subchannel."

Subclause 3.2.3

Model: In paragraph b), the last sentence is replaced by: "represented by a <channel connection> where the <block identifier> or **env** has been replaced by the appropriate new channel."

Subclause 3.3

Concrete textual grammar: The following text is added to the subclause:

> "Within each *gate-boundary-delimited part* (this term is explained in 7.3.4) of a specification, all channels and all signal lists on channels must be explicit if the part uses signals at different refinement levels."

Subclause 4.13

Concrete textual grammar: The 1st paragraph after the grammar rules is replaced by:

> "The use of **nodelay** is described in 7.3.4 and 7.3.5."

The 4th paragraph after the grammar rules is replaced by:

> "A remote variable mentioned in an <import expression> must be in the complete output set (7.3.1) of an enclosing process type, process set, service type or service."

Subclause 4.13

Model: In the 1st paragraph, the first two sentences are replaced by:

> "An import operation is modelled by exchange of implicitly defined signals."

After the 3rd paragraph, the following one is added:

> "On each channel or signal route mentioning the remote variable, the remote variable is replaced by *xQUERY*. For each such channel or signal route, a new channel or signal route is added in the opposite direction; this channel or signal route carries the signal *xREPLY*. In case of a channel, the new channel has the same delaying property as the original one."

Subclause 4.14

Concrete textual grammar: The 1st paragraph after the grammar rules is replaced by:

> "The use of **nodelay** is described in 7.3.4 and 7.3.5."

The 4th paragraph after the grammar rules is replaced by:

> "A remote procedure mentioned in a <remote procedure call> must be in the complete output set (7.3.1) of an enclosing process type, process set, service type or service."

Subclause 4.14

Model: In the 1st paragraph, the first two sentences are replaced by:

> "A remote procedure call is modelled by exchange of implicitly defined signals."

After the 2nd paragraph, the following one is added:

> "On each channel or signal route mentioning the remote procedure, the remote procedure is replaced by *pCALL*. For each such channel or signal route, a new channel or signal route is added in the opposite direction; this channel or signal route carries the signal *pREPLY*. In case of a channel, the new channel has the same delaying property as the original one."

Subclause 6.1.1.3

Semantics: The 2nd paragraph is deleted.

Subclause 6.1.1.4

Semantics: The 2nd paragraph is deleted.

Subclause 6.1.4

Concrete textual grammar: In the 3rd paragraph, line 4, the text "if the set of signals" is changed to "if the set of signals (if specified)".

The 4th and 5th paragraphs are replaced by:

> "If the type denoted by <base type> in a <textual typebased block definition> or <textual typebased process definition> contains signal routes, the following rule applies: For each combination of (gate, signal, direction) defined by the type, the type must contain at least one signal route which – for the given direction – mentions **env** and the gate and either mentions the signal or has no explicit <signal list> associated. In the latter case, it must be possible to derive that the signal route is able to carry the signal in the given direction.
>
> If the type contains signal routes mentioning remote procedures or remote variables, a similar rule applies.
>
> If a <block substructure definition> in the block type denoted by <base type> in a <textual typebased block definition> contains channels, the following rule applies: For each combination of (gate, signal, direction) defined by the block type, the type must contain at least one channel which – for the given direction – mentions **env** and the gate and either mentions the signal or has no explicit <signal list> associated. In the latter case, it must be possible to derive that the channel is able to carry the signal in the given direction.
>
> If the block type contains channels mentioning remote procedures or remote variables, a similar rule applies.
>
> It is allowed to use explicit communication paths for "normal" signals, but implicit communication paths for remote procedures and remote variables, even though the specification explicitly defines and/or uses gates which mention remote procedures or remote variables.
>
> It is allowed to specialize the implicit gate rpv_gate explicitly in a subtype. An explicit specialization of rpv_gate is handled like a specialization of any other gate (for example, it is allowed to add "normal" signals to rpv_gate).
>
> In a subtype of a type with explicit communication paths for signals, but implicit communication paths for remote procedures and remote variables, it is allowed to add explicit <channel path>s or <signal route path>s which do not contain explicit <signal list>s, or which mention remote procedures or remote variables."

Subclause 6.1.4

Model: The following text is added at the beginning:

> "*Implicit gates for remote procedures and remote variables*: An implicit gate called rpv_gate is derived for each block type, process type and service type which (directly or through contained instances) exports remote procedures or remote variables, and which does not have any explicit gates mentioning these remote procedures or remote variables. The details of this are described in 7.3.2.
>
> After this transformation, each exported or imported remote procedure or remote variable must be mentioned on at least one gate (in the proper direction) of the type."

The title "*Transformation of gates:*" is added at the beginning of the 1st paragraph of the existing text.

Subclause 7

A new subclause is added:

7.3 Insertion of implicit gates, channels, signal routes and signal lists

The transformations below are applied in the given order.

7.3.1 Complete input and output sets

Complete <valid input signal set>s: The complete <valid input signal set> of a <process type definition>, <process definition>, <service type definition> or <service definition> is the set of signals, remote procedures and remote variables which:

- occur in the complete <valid input signal set> of the supertype, if any, of the process or service type (in case of a <process type definition> or <service type definition>); or
- are mentioned on the incoming gates of the process or service type (in case of a <process type definition> or <service type definition>), or are mentioned explicitly on signal routes leading to the process set or service (in case of a <process definition> or <service definition>); or
- are exported by the procedure definitions and variable definitions in the process type, process set, service type or service (only for remote procedures and remote variables).

In case of a <process type definition> or <process definition> containing services, the complete <valid input signal set> furthermore contains the signals, remote procedures and remote variables which are contained in the complete <valid input signal set>s of these services.

Complete input and output sets: The complete input set of a <process type definition>, <process definition>, <service type definition> or <service definition> is its complete <valid input signal set>. The complete output set of a <process type definition>, <process definition>, <service type definition> or <service definition> is the set of signals, remote procedures and remote variables which:

- occur in the complete output set of the supertype, if any, of the process or service type (in case of a <process type definition> or <service type definition>); or
- are mentioned on the outgoing gates of the process or service type (in case of a <process type definition> or <service type definition>), or are mentioned explicitly on signal routes leading from the process set or service (in case of a <process definition> or <service definition>); or
- are mentioned in the output nodes of the process set or service (in case of a <process definition> or <service definition>) (only for signals); or
- are mentioned in the imported-procedure specifications and imported-variable specifications in the process type, process set, service type or service (only for remote procedures and remote variables).

In case of a <process type definition> or <process definition> containing services, the complete output set furthermore contains the signals, remote procedures and remote variables which are contained in the complete output sets of these services.

NOTE – The complete input and output sets described here do not include timer signals or implicitly defined signals.

7.3.2 Implicit gates

Implicit gates for remote procedures and remote variables: An implicit <gate definition> is added to each block type, process type and service type which:

- directly or through contained instances, exports or imports remote procedures or remote variables; and
- does not have explicit gates mentioning these remote procedures or remote variables.

The gate denoted by this <gate definition> has the name rpv_gate, has an incoming part if the type exports at least one remote procedure or remote variable, and has an outgoing part if the type imports at least one remote procedure or remote variable. Each part mentions the remote procedures and remote variables which are communicated in the corresponding direction.

The implicit gate rpv_gate (if present) and the implicit communication paths specially for remote procedures and remote variables, are added to the supertype before the contents of the supertype and the subtype are merged.

If the type inherits such an implicit gate from another type and exports or imports new remote procedures or remote variables, the new implicit <gate definition> contains the keyword **adding** and mentions the new remote procedures and remote variables.

7.3.3 Implicit channels and signal routes

In the description below, it is understood that when implicit channels or signal routes are derived, implicit **connect** constructs are also derived in the cases where this is appropriate.

Implicit channels: If a <system definition> or <system type definition> contains no explicit <channel definition>s, implicit <channel definition>s without <signal list>s are derived such that each possible pair of "channel endpoints" are connected in each possible direction. A "channel endpoint" here means one of the following:

- A <block definition>.
- A gate of a <textual typebased block definition>.
- The system environment.

However, no implicit <channel definition> connects a <block definition> with itself; instead, a block denoted by a <block definition> communicates with itself via *internal* implicit communication paths.

Thereafter, if a <block substructure definition> (of a <block definition> or <block type definition>) or <channel substructure definition> contains no <channel definition>s, implicit <channel definition>s are derived in a similar way. Here, a "channel endpoint" means the same as above, or one of the following:

- A(n outside) channel connected to the enclosing scope unit (in case of a <block definition>).
- A gate of the enclosing scope unit (in case of a <block type definition>).
- The "enclosing" channel (in case of a <channel substructure definition>).

Nested substructures are thereafter handled recursively in the same way.

All implicit channels are unidirectional and delaying.

Implicit signal routes: If a <block definition> or <block type definition> contains no explicit <signal route definition>s, implicit <signal route definition>s without <signal list>s are derived such that each possible pair of "signal-route endpoints" are connected in each possible direction. A "signal-route endpoint" here means one of the following:

- A <process definition>.
- A gate of a <textual typebased process definition>.
- A channel connected to the enclosing scope unit (in case of a <block definition>).
- A gate of the enclosing scope unit (in case of a <block type definition>).
- The system environment (in case of a <block definition> defined as a system).

However, no implicit <signal route definition> connects a <process definition> with itself; instead, a process set denoted by a <process definition> communicates with itself via *internal* communication paths (if decomposed into services) or without communication paths at all (if not decomposed into services).

All implicit signal routes are unidirectional.

Thereafter, if a <process definition> or <process type definition> contains services, but no signal routes, implicit <signal route definition>s are derived in a similar way.

Implicit channels and signal routes for remote procedures: If a scope unit contains explicit channels or signal routes for signals, but no explicit channels or signal routes for remote procedures and remote variables, implicit channels/signal routes are derived for remote procedures and remote variables only, in the scope unit in question. Subclause 7.3.4 describes the precise conditions for when and how this happens.

7.3.4 Implicit channels and signal routes for remote procedures and remote variables

In the following, the term *gate-boundary* is convenient. A gate-boundary is the boundary of a <system type definition>, <block type definition>, <process type definition> or <service type definition>.

An SDL specification thus consists of parts delimited by gate-boundaries. These parts are called *gate-boundary-delimited* parts in the following.

Within each gate-boundary-delimited part, implicit channels or signal routes are derived specially for remote procedures or remote variables if the following conditions are fulfilled (i.e. within each gate-boundary-delimited part, remote procedures and remote variables are treated the same way as "normal" signals unless the following conditions are fulfilled):

- All <channel path>s and <signal route path>s (there may be zero) in the gate-boundary-delimited part contain explicit <signal list>s.
- No <signal list> in the <channel path>s or <signal route path>s in the part mentions a remote procedure or remote variable.

In this case, any implicit channels and signal routes in the affected scope units are derived in a way similar to the one described in 7.3.3 and 7.3.5, but with the following modification: Each implicit <channel path> or <signal route path> derived according to the rules in 7.3.3, is replaced by two or three <channel path>s or <signal route path>s. The <signal list> of the "original" implicit <channel path> or <signal route path> and derived according to the rules in 7.3.5, is split among the two or three resulting <channel path>s or <signal route path>s as follows:

- One path is only filled in with "normal" signals. If the path is a <channel path>, it is delaying. This path is *not* derived if the containing scope unit already contains channels or signal routes for "normal" signals.
- One path is only filled in with remote procedures and remote variables which have been defined *without* the **nodelay** attribute. If the path is a <channel path>, it is delaying.
- One path is only filled in with remote procedures and remote variables which have been defined *with* the **nodelay** attribute. If the path is a <channel path>, it is *not* delaying.

7.3.5 Implicit signal lists on channels and signal routes

Implicit signal lists on channels and signal routes: The specification now contains a number of sequences of <channel path>s and <signal route path>s where each sequence has the following properties:

- Each <channel path> or <signal route path> (except the last one) in the sequence is connected to the next <channel path> or <signal route path>.
- None of the <channel path>s or <signal route path>s in the sequence contain <signal list>s.
- The originating "endpoint" of the first <channel path> or <signal route path> is one of the following:
  - a number of <channel path>s or <signal route path>s some of which have explicit <signal list>s;
  - a gate;
  - a <process definition> not containing services;
  - a <service definition>;
  - the system environment.

The destination "endpoint" of the last <channel path> or <signal route path> satisfies a similar condition.

The <signal list>s in the <channel path>s and <signal route path>s in the sequences are now filled in such that they are able to convey "as many" signal types, remote procedure calls and import requests as possible. This derivation uses the complete input and output sets of any <process definition>s and <service definition>s at the "endpoints" of the sequences.

That is, each sequence contributes with the intersection of the following sets of signals, remote procedures and remote variables, to all <signal list>s in the sequence:

- The originating set of the sequence (disregarded if the originating "endpoint" is the system environment).
- The destination set of the sequence (disregarded if the destination "endpoint" is the system environment).
- The set of signals, remote procedures and remote variables which are visible along the whole sequence.

When filling in a remote procedure or remote variable on a given channel, the delay/**nodelay** property of the remote procedure or remote variable is ignored unless the channel was derived according to the rules in 7.3.4. If a <channel path> or <signal route path> was derived according to the rules in 7.3.4, the <signal list> on this path is filled in according to the additional rules in 7.3.4.

Now all implicit <channel path>s and <signal route path>s with empty signal lists are removed. Any **connect** constructs which become "orphaned" this way, are also removed.

10 Using a block or a process or a service as a system

The SDL view of an SDL system is that it is a self-contained entity which communicates with its environment by means of signals. However, often more than one view exists. Consider for example a computer. It can be regarded as a well-defined system on its own, but it can also be a component of a larger system (for example a computer network). It is therefore useful to allow a service, a process or a block to be considered a system on its own. This feature is also useful for simple systems consisting of, for example, one process.

A <system definition> can thus consist of only one (possible type-based) block, process or service. Note that a separate extension described elsewhere enables a scope unit to have a <use clause> which means that such a <system definition> still can be based on entities defined in a package.

Any signal route/channel connections are left out for the outermost process/block level.

10.1 Changes

Subclause 2.4.1.1

Concrete grammar add: <package list> and <system definition> cannot both be omitted. The <package list> can only be specified if the <system definition> is a <textual system definition> or a <system diagram>.

Subclause 2.4.1.3

Change production <system definition> to:

<system definition> ::=

<textual system definition>

| <system diagram>

| <block definition>

| <block diagram>

| <textual typebased block definition>

| <graphical typebased block definition>

| <process definition>

| <process diagram>

| <textual typebased process definition>

| <graphical typebased process definition>

| <service diagram>

| <textual typebased service definition>

| <graphical typebased service definition>

Subclause 2.4.1.3

Add to *Model:*

1) A <system definition> being a <service definition> or a <textual typebased service definition> is derived syntax for a <process definition> having the same name as the service, containing implicit signal routes and containing the <service definition> or <textual typebased service definition> as the only definition.

   A <system definition> being a <service diagram> or a <graphical typebased service definition> is derived syntax for a <process diagram> having the same name as the service, containing implicit signal routes and containing the <service diagram> or <graphical typebased service definition> as the only definition.

2) A <system definition> being a <process definition> or a <textual typebased process definition> is derived syntax for a <block definition> having the same name as the process, containing implicit signal routes and containing the <process definition> or <textual typebased process definition> as the only definition.

   A <system definition> being a <process diagram> or a <graphical typebased process definition> is derived syntax for a <block diagram> having the same name as the process, containing implicit signal routes and containing the <process diagram> or <graphical typebased process definition> as the only definition.

3) A <system definition> being a <block definition> or a <textual typebased block definition> is derived syntax for a <system definition> having the same name as the block, containing implicit channels and containing the <block definition> or <textual typebased block definition> as the only definition.

   A <system definition> being a <block diagram> or a <graphical typebased block definition> is derived syntax for a <system diagram> having the same name as the block, containing implicit channels and containing the <block diagram> or <graphical typebased block definition> as the only definition.

For the following two conditions the contents of substructure contained in a <block diagram> or <block definition> is considered directly contained in the <block diagram> or <block definition>:

A <channel definition area> or a <signal route definition area> for a channel or signal route directly contained in a <system definition> cannot be *associated with* <channel identifiers> or <external signal route identifiers>.

The scope unit directly contained in a <system definition> cannot contain <channel to route connection> s, <channel connection>s or <signal route to route connection>s.

11 State expression

11.1 Motivation

In conjunction with use of asterisk state, it is often needed to have access to information about the identity of the most recently entered state. Such a feature is part of SDL-96.

11.2 Solution

A new imperative operator: **state** has been introduced. It denotes the Character string value containing the spelling of the most recently entered state.

11.3 Example

state *; /* Say covering among others the state named state1 */

input *;

...

decision state;

(state1) : **task** 'do something for state1';

else : **task** 'do something else';

enddecision;

11.4 Changes

Subclause 5.4.4

Add an alternative in <imperative operator>

<imperative operator> ::=

<now expression>

| ...

| <anyvalue expression>

| <state expression>

Add a new **subclause 5.4.4.7**:

Concrete textual grammar

<state expression> ::=

state

Model

<state expression> is derived syntax for a Charstring literal which contains the lowercase spelling of the name of the most recently entered state of the nearest enclosing scope unit. If there is no such state, <state expression> denotes the empty string (''). The construct is transformed together with <dash nextstate> (see 7.1 step 18).

12 Extended use of packages

12.1 Motivation

Currently, packages can only be used in packages and on the system level. The disadvantage of this is that all definitions in a package will be visible in the whole system scope. Often there is a need to include definitions at a lower level. For example, it is often the case that some data type is only needed in a single process in the system, but this data type is anyway so general that it may be reused in other systems. Therefore, there is a need to be able to use packages at the scope unit where this is needed. Such a change will also make an integration of SDL and GDMO easier, as GDMO allows the use of packages on managed object class level (which would correspond to SDL process type level).

12.2 Solution

Allowing the use of packages in other scope units than system, like block/block types, process/process types, service/service types, etc.

12.3 Changes

Page 18, paragraph starting with: "It is allowed ...": after the sentence ending with "is the same as the rightmost part of the full <qualifier> denoting this scope unit." add a new sentence:

> With respect to visibility and use of qualifiers a <package reference clause> associated to a scope unit S is regarded as representing a package definition directly enclosing S and defined in the scope unit where S is defined.
>
> NOTE – In the concrete syntax, packages cannot be defined inside other scope units. The above rule is only for defining the visibility rules which apply for packages. A consequence of this rule is that names in a package can be referred to using different qualifiers, one for each enclosed <package reference clause> of the package.

Page 24, 3rd sentence, replace: "Definitions within a package are made visible to a system or other packages" by: "Definitions within a package are made visible to another scope unit".

Page 25, last sentence before "1)" and "2)", replace: "another <package> or the <system definition>" by "a scope unit"

Page 25, bullet 1), replace: "The <package> or <system definition> has a <package reference clause> mentioning the <package and" by:

> "The scope unit has a <package reference clause> associated, or any scope unit enclosing the scope unit has a <package reference clause> associated and the <package reference clause> mentions the <package> and"

Page 26: delete the paragraph starting with "Names having their defining occurrence in a <package> are referred ..."

Page 26, *Model*, add at the bottom:

> NOTE – When a type is specialized (see 6.3), any <package reference clause> associated to the supertype is not copied to the specialized type.

12.3.1 General changes to the concrete textual grammar

Add "{<package reference clause>}*" at the beginning of the right-hand sides in the definitions of <block definition>, <process definition>, <service definition>, <procedure definition>, <operator definition>, <block substructure definition>, <channel substructure definition>, <system type definition>, <block type definition>, <process type definition>, and <service type definition>.

12.3.2 General changes to the concrete graphical grammar

Add "{<package reference area>} *is associated with*" at the beginning of the right-hand sides in the definitions of <block diagram>, <process diagram>, <service diagram>, <procedure diagram>, <operator diagram>, <block substructure diagram>, <channel substructure diagram>, <system type diagram>, <block type diagram>, <process type diagram>, and <service type diagram>.

Add a sentence "The <package reference area> must be placed on the top of the system frame symbol." at the clauses where the above-mentioned diagrams are defined.

13 Operators with zero arguments in the ***operators*** part of a newtype

13.1 Motivation

The original purpose of operators with zero arguments (i.e. literals) was probably to use them to contribute to the value set of a sort (e.g. for defining "enumerated values"). A facility such as **ordering** indicates this.

A possibility in SDL-92 is to use synonyms instead, but this has the following drawbacks:

- Synonyms cannot be overloaded.
- The "signatures" of the synonyms are defined in another place of the specification than the signatures of the related non-nullary operators ("outside" and "inside" the **newtype**, respectively). If the **newtype** has context parameters, this is actually impossible to define the synonym outside the **newtype**.
- The "behaviour" of the synonyms is defined in another place of the specification than the behaviour of the related non-nullary operators. Firstly, synonyms are defined "outside" the **newtype**, while axioms or algorithmic operator definitions are given "within" the **newtype**. Secondly (in the case of algorithmic operators), the behaviour of the non-nullary operators may be defined in **referenced** operator diagrams – i.e. again separated from the "behaviour" of the related synonyms.
- Synonyms cannot be used in axioms.

13.2 Solution

In SDL-96, operators with zero arguments are therefore allowed in the **operators** part of a **newtype**. As opposed to the nullary operators in the **literals** part, nullary operators in the **operators** part are not influenced by facilities such as **ordering**. On the contrary, the same facilities are available for nullary operators in the **operators** part as for all other operators – e.g. the possibility of defining them algorithmically or externally and the possibility to define "new" nullary operators of "old" types (e.g. Pi of type Real).

13.3 Example

```
newtype Color

    literals Yellow, Blue, Green, Red;

    operators

        ordering;

        First: -> Color; /* not allowed  SDL-92  */

        Last: -> Color; /* not allowed  SDL-92 */

        /* other operators */

    axioms

        First == Yellow;

        Last == Red;

        /* other axioms */

endnewtype;
```

the literals Yellow, Blue, Green, and Red constitute the value set of Color, and the four literals are the ones ordered by **ordering**. Today, First and Last also have to be defined under **literals**, and the ordering thus has to be defined in a much more complicated way (for the user) than just stating **ordering** under **operators**.

13.4 Impact on Recommendation Z.105

The extension enables a more user-friendly access to the operators First and Last defined for the predefined Enumeration type in Recommendation Z.105. As operators in SDL-92 are not allowed to have zero arguments, those two operators must be given a 'dummy' argument when used.

For the Z.105 version which corresponds to SDL-96, the two operators are overloaded:

```
First: -> Enumeration.

Last: -> Enumeration.
```

13.5 Changes

Subclause 5.2.2

Subclause title: The title of the subclause is changed to "**Literals and operators**".

Subclause 5.2.2

Abstract grammar: The grammar rules for *Signature*, *Literal-signature*, *Operator-signature* and *Argument-list* are replaced by:

Signature :: Operator-name

Argument-list

Result

*Argument-list = Sort-reference-identifier**

The grammar rule for *Literal-operator-name* is deleted.

Subclause 5.2.2

Concrete textual grammar: The grammar rule for <literal signature> is replaced by:

<literal signature> ::=

<literal name> |

<name class literal>

<literal name> ::=

<literal operator name> |

<extended literal name>

The grammar rule for <argument list> is changed to:

<argument list> ::=

[<argument sort> { , <argument sort> }*]

The whole text after the grammar rules is replaced by:

"The alternatives <name class literal>, <extended literal name>, <ordering>, <noequality>, <extended operator name>, <generator sort> and <syntype> are not part of the data kernel.

A *Signature* is represented by a <literal signature> which contains a <literal name> not being a <generator formal name>, or by an <operator signature> which contains an <operator name> not being a <generator formal name>.

If the *Signature* is represented by a <literal signature>, the *Argument-list* is empty, and the *Result* is the sort introduced by the <partial type definition> defining the <literal signature>. If the *Signature* is represented by an <operator signature>, each *Sort-reference-identifier* in the *Argument-list* is represented by an <argument sort> in the <argument list>, and the *Result* is represented by the <result>.

The other possible forms of <literal signature> and <operator signature> are shorthands. Each of these shorthands corresponds in general to several *Signature*s.

A <literal name> or <operator name> not being a <generator formal name> corresponds to an *Operator-name*. This *Operator-name* is unique within the defining scope unit even though the corresponding <literal name> or <operator name> may not be unique.

The unique *Operator-name* is derived from:

1) the <literal name> or <operator name>; plus
2) the (possibly empty) list of argument sort identifiers; plus
3) the result sort identifier; plus
4) the sort identifier of the partial type definition in which the <literal name> or <operator name> is defined.

Wherever a <literal identifier> or <operator identifier> is specified, the unique *Operator-name* in *Operator-identifier* is derived in the same way with the list of argument sorts and the result sort derived form context. Two operators (of which each can be a literal or a "normal" operator) with the same name, but differing by one or more of the argument or result sorts, have different *Operator-name*s.

Wherever a <qualifier> of a <literal identifier> or <operator identifier> contains a <path item> with the keyword **type**, the <sort name> after this keyword does not form part of the *Qualifier* of the *Operator-identifier*, but is used to derive the unique *Name* of this *Identifier*. In this case the *Qualifier* is formed from the list of <path item>s preceding the keyword **type**."

Subclause 5.2.2

Semantics: The 3rd and 4th paragraphs are changed to:

"An operator (with no arguments) defined in a <literal signature> is called a literal. An operator with no arguments and defined in an <operator signature> is called a nullary operator.

A literal or a nullary operator represents a fixed value belonging to the result sort of the operator."

The following new paragraph is inserted immediately before the last one:

"NOTE 1 – From the point of view of the abstract grammar, there is no difference between literals and nullary operators. The difference lies in how the two are used in SDL: Certain facilities, e.g. **ordering**, only apply to literals, not to nullary operators. Nullary operators are handled like operators with one or more arguments – for example, their behaviour can be defined by means of <operator definition>s."

The existing Note in the 1993 version is changed to:

"NOTE 2 – guideline: An <operator signature> should *normally* mention the sort introduced by the enclosing <partial type definition> as either an <argument sort> or a <result>. The exception from this guideline is the case where a <partial type definition> is just used as a "container" ("mini-package") to group "new" operators on "old" sorts together when these operators are related in some way. The <sort name> defined by such a <partial type definition> should only be used in <qualifier>s of the defined operators."

The following new paragraph is added at the end:

"NOTE 3 – guideline: An operator (with no arguments) which is used to contribute to the value set of a sort, should be defined as a literal. An operator (with no arguments) which is used to define a name for an "existing" value, should be defined as a nullary operator. For example, this guideline has been used to define the following enumerated sort:

```
newtype Color

    literals

        Red, Green, Blue, Yellow;

    operators

        ordering;

        First: -> Color;

        Last: -> Color;

        /* other operators */

    axioms

        First == Red;

        Last  == Yellow;

        /* other axioms */

endnewtype Color;
```

"

Subclause 5.2.3

Abstract grammar: The grammar rule for *Ground-term* is changed to:

*Ground-term :: Operator-identifier Ground-term** |

Conditional-ground-term

The grammar rule for *Literal-operator-identifier* is deleted.

Subclause 5.2.3

Concrete textual grammar: The grammar rule for <ground term> is changed to:

<ground term> ::=

<literal operator identifier> |

<operator identifier> ([<ground term>] { , <ground term> }*) |

(<ground term>) |

<extended ground term>

In the paragraph starting with "c)", "*literal operator identifier*" is changed to "*Operator-identifier*", and "literal" is replaced by "literal or operator".

In the 4th-last paragraph, "<operator name>" is changed to "<literal name> or <operator name>" (two occurrences), and "operator" is changed to "literal or operator" (three occurrences).

In the penultimate paragraph, "*operator identifier* or *literal operator identifier*" is changed to "*Operator-identifier*".

The last paragraph is changed to:

"NOTE – guideline: An axiom should be relevant to the literals and operators of the enclosing partial type definition by mentioning a literal or operator whose signature is defined within the partial type definition. A given axiom should occur only once."

Subclause 5.3.1

Concrete textual grammar: The grammar rule for <extended ground term> is changed to:

<extended ground term> ::=

<extended literal identifier> |

<extended operator identifier> ([<ground term>] { , <ground term> }*) |

<ground term> <infix operator> <ground term> |

<monadic operator> <ground term> |

<conditional ground term>

The alternative <name class literal> is deleted from the grammar rule for <extended literal name>.

Subclause 5.3.1.2

Concrete textual grammar: In the 2nd paragraph after the grammar rules, "*Literal-operator-identifier*" is changed to "*Operator-identifier*". In the 3rd paragraph, "*Literal-operator-name*" is changed to "*Operator-name*".

Subclause 5.3.1.7

Abstract grammar: In the last paragraph, "*literal operator identifier*" is changed to "*Operator-identifier* corresponding to a literal".

Subclause 5.3.1.7

Model: The following paragraph is added at the end:

"NOTE – The keyword **ordering** does not affect nullary operators defined in <operator signature>s."

Subclause 5.3.1.11

Model: The following paragraph is added at the end:

"NOTE 2 – Inherited literals are renamed by means of <literal renaming>. Inherited nullary operators are handled by the keyword **all** or are renamed by means of <inheritance list>, just like all other inherited "normal" operators."

(Editor's note – The existing Note in the 1993 version becomes NOTE 1.)

Subclause 5.3.1.15

Model: The following paragraph is added at the end:

"NOTE – Literal mappings do not affect nullary operators defined in <operator signature>s."

Subclause 5.3.2

Concrete textual grammar: In the grammar rule for <operator definition>, "<formal parameters> <end>" is changed to "[<formal parameters> <end>]". In the grammar rule for <textual operator reference>, "<formal parameters>" is changed to "[<formal parameters>]".

In the 3rd paragraph after the grammar rules, the text "(if present)" is added after "<formal parameters>".

In the 5th paragraph, "<formal parameters>" is replaced by "[<formal parameters>]" (two occurrences).

Subclause 5.3.2

Concrete graphical grammar: In the grammar rule for <operator heading>, "<formal parameters>" is replaced by "[<formal parameters>]".

Subclause 5.3.2

Model: The following paragraph is added at the end:

"NOTE – It is not possible to specify an <operator definition> for a <literal signature>."

Subclause 5.3.3.2

Concrete textual grammar: The grammar rule for <ground primary> is changed to:

<ground primary> ::=

<literal identifier> |

<operator identifier> ([<ground expression list>]) |

(<ground expression>) |

<conditional ground expression>

14 Reformulation of subclause 6.3.2 concerning virtuals

A block type, process type, service type or procedure may be specified as a virtual type when it is defined locally to another type (denoted as the *enclosing* type in this subclause). A virtual type may be redefined in specializations of the enclosing type.

Concrete textual grammar

<virtuality> ::=

virtual | **redefined** | **finalized**

<virtuality constraint> ::=

atleast <identifier>

<virtuality> and <virtuality constraint> are part of a type definition. See 6.1.1.2 (block type), 6.1.1.3 (process type), 6.1.1.4 (service type) and 2.4.6 (procedure).

A virtual type is a type having **virtual** or **redefined** as <virtuality>. A redefined type is a type having **redefined** or **finalized** as <virtuality>.

Every virtual type has associated a virtuality constraint which is an <identifier> of the same entity kind as the virtual type. If <virtuality constraint> is specified, the virtuality constraint is the contained <identifier>, otherwise the virtuality constraint is derived as described below.

A virtual type and its constraints cannot have formal context parameters.

Only virtual types may have <virtuality constraint> specified.

If <virtuality> is present both in the reference and the referenced definition, then they must be equal. If <procedure preamble> is present in both the procedure reference and in the referenced definition, they must be equal.

A virtual type must have exactly the same formal parameters, gates and signals on its gates as its constraint.

Semantics

A virtual type may be redefined ...

Accessing a virtual type by means of a ...

A virtual or redefined type which has no <specialization> specified, may have an implicit <specialization> . The virtuality constraint and the possible implicit <specialization> is derived as follows:

Given a virtual type V with a <virtuality constraint> VA and a specialization VS and given a redefined type R of V with <virtuality constraint> RA and a <specialization> RS, then the following holds (all rules are applied and in the given order):

- if VA is omitted, then VA is the same as V;
- if VS is omitted and VA does not denote V then VS is the same as VA;
- if VS is present, then VS must be the same as or a subtype of VA;
- if RA is omitted then RA is the same as R;
- if RS is omitted then RS is the same as VA;
- RA must be the same as or a subtype of VA;
- RS must be the same as or a subtype of RA;
- if R is a virtual type, the same rules apply for R as for V.

A subtype of a virtual type is a subtype of the original virtual type and not of a possible redefinition.

15 Maintenance of SDL

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

15.1 Terminology

15.1.1 An **error** is an internal inconsistency within Recommendation Z.100.

15.1.2 A **textual correction** is a change to text or diagrams of Recommendation Z.100 which corrects clerical or typographical errors.

15.1.3 An **open item** is a concern identified but not resolved. An open item may be identified either by a Change Request, or by agreement of the Study Group or Working Party.

15.1.4 A **deficiency** is an issue identified where the semantics of SDL are not (clearly) defined by Recommendation Z.100.

15.1.5 A **clarification** is a change to the text or diagrams of Recommendation Z.100 which clarifies previous text or diagrams which could be ambiguously understood without the clarification. The clarification should attempt to make Recommendation Z.100 correspond to the semantics of SDL as understood by the Study Group or Working Party.

15.1.6 A **modification** is a change to the text or diagrams of Recommendation Z.100 which changes the semantics of SDL.

15.1.7 A **decommitted feature** is a feature of SDL which is to be removed from SDL in the next revision of Recommendation Z.100.

15.1.8 An **extension** is a new feature, which must not change the semantics of features defined in Recommendation Z.100.

15.2 Rules for maintenance

In the following text references to Recommendation Z.100 shall be considered to include this Addendum and Recommendation Z.105.

1) When an error or deficiency is detected in Recommendation Z.100, it must 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 Z.100 and come into effect immediately.

2) Except for error corrections and resolution of open items from the previous study period, modifications and extensions to SDL may only be considered as the result of a request for change 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 SDL is unsuitable.

3) Modifications and extensions not resulting from error correction shall be widely publicised and the views of users and tool-makers 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 Z.100.

4) Until a revised Recommendation Z.100 is published, a Master list of Changes to Recommendation Z.100 will be maintained covering Recommendation Z.100 and all annexes except the formal definition. Appendices, Addenda or Supplements will be issued as decided by the Study Group. To ensure effective distribution of the Master list of Changes to Recommendation Z.100, it will be published as COM Reports and by appropriate electronic means.

5) For deficiencies in Recommendation Z.100 the formal definition should be consulted. This may lead to either a clarification or correction, that is recorded in the Master list of changes to Recommendation Z.100.

15.3 Change request procedure

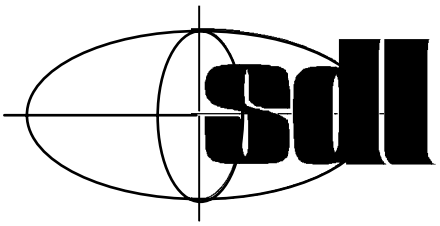
The change request procedure is designed to enable SDL users from within and outside ITU-T to ask questions about the precise meaning of Recommendation Z.100, make suggestions for changes to SDL or Recommendation Z.100, and to provide feedback on proposed changes to SDL. Proposed changes to SDL will be published by the SDL experts' group 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, 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.

All change requests will be treated by meetings of the ITU-T Study Group responsible for Recommendation Z.100. For corrections or clarifications the changes may be put on the list of corrections without consulting users. Otherwise a list of open items is compiled. The information should be distributed to users:

- as ITU-T white contribution reports;
- via the SDL Newsletter (ISSN 1023-7151);
- electronic mail to SDL mailing lists (such as "SDL_News");
- others means as agreed by the Study Group 10 experts.

Reactions from users will be evaluated by Study Group 10 experts to determine the level of support and opposition for each change. 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 Z.100. Users should be aware that until changes have been incorporated and approved by the Study Group responsible for Recommendation Z.100 they are not Recommended by ITU-T.



Change Request Form

Please fill in the following details		
Character of change:	q error correction q simplification q modification	q clarification q extension q decommission
Short summary of change request		
Short justification of the change request		
Is this view shared in your organisation	q yes	q no
Have you consulted other users	q yes	q no
How many users do you represent?	q 1-5 q 11-100	q 6-10 q over 100
Your name and address		

please attach further sheets with details if necessary

SDL (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.ch

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
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	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
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
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 and open system communication
Series Z	**Programming languages**