International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.106

(10/2019)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

Specification and Description Language – Common interchange format for SDL 2010

Recommendation ITU-T Z.106

International Telecommunication Union

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

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

Recommendation ITU-T Z.106

Specification and Description Language – Common interchange format for SDL-2010

Summary

Recommendation ITU-T Z.106 defines the common interchange format of Specification and Description Language (SDL-CIF). The SDL-CIF is intended for the interchange of graphical SDL-2010 specifications (SDL-GR) made on different tools that do not use the same storage format. Prior to the definition of SDL-CIF, the textual phrase representation of SDL-2010 (SDL-PR) was used to interchange specifications with the disadvantage that all graphical information was lost, making the same specifications often look very dissimilar in different environments. With the SDL-CIF, this disadvantage is reduced to a minimum, as it contains most of the graphical information. The SDL-CIF improves the independence from specific tool vendors and allows standards bodies to accept specifications in SDL-CIF irrespective of the tool they use for their internal work. This also improves productivity by allowing specifications to be made on the accustomed tool. All SDL-2010 tool vendors are encouraged to provide facilities for importing and exporting SDL-CIF.

This Recommendation defines how SDL-2010 descriptions are stored in order to be interchanged between tools coming from different vendors. It does not take into account the message sequence chart (MSC) notation. SDL-CIF is an optional part of SDL-2010. SDL-CIF is based on the SDL-PR syntax, the textual phrase representation of SDL-2010 also defined in this Recommendation. SDL-CIF is readable and written by tools as well as users. All the constructs available in SDL-2010 are able to be expressed in graphical form or in the purely textual SDL-PR form. Constraints on graphical presentation are expressed in SDL-CIF by adding specific annotations to SDL-PR. As a result, most SDL-PR descriptions are legal SDL-CIF descriptions. SDL-CIF is an open storage format as it includes a mechanism of tool-specific directives. This mechanism allows an SDL-CIF-compliant tool to extend the format by adding specific information. SDL-CIF is also easily implemented and provides tool vendors with two levels of tool conformance and concepts of mandatory and optional directives.

SDL-PR is an alternative text-only syntax for the Specification and Description Language. Before 2002, SDL-PR was published as part of ITU-T Z.100, but as the main use of this notation is for communication within and between tools the definition has been moved to this Recommendation. SDL-PR is Level 0 SDL-CIF and allows the interchange of syntactically complete SDL-2010 descriptions, usually as a single file per system. Conformance to SDL-PR requires the model to conform to the corresponding semantics defined in Recommendations ITU-T Z.101, ITU-T Z.102, ITU-T Z.103, ITU-T Z.104, ITU-T Z.105 and ITU-T Z.107.

This Recommendation introduces two further levels of SDL-CIF. Two further conformance levels are defined, one at a more liberal SDL-PR level and the second including graphical information. The complete grammar is described with the related semantics. Mandatory and optional directives are described, as well as the format for tool-specific directives. Current tool-specific directives are described in Appendix I.

Two levels of SDL-CIF conformance are defined as level 1 and level 2. Level 1 is very close to SDL-PR, but it supports incomplete descriptions in SDL-2010. Level 2 includes level 1 and is able to capture most of the graphical information of SDL-GR diagrams. An SDL-CIF specification shall identify which of these two levels it complies with. Similarly, tool vendors that use the SDL-CIF should also identify the SDL-CIF level they comply with for their import and export functions.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T Z.106	1996-10-18	10	11.1002/1000/3918
2.0	ITU-T Z.106	2000-11-24	10	11.1002/1000/5240
3.0	ITU-T Z.106	2002-08-06	17	11.1002/1000/6030
4.0	ITU-T Z.106	2011-12-22	17	11.1002/1000/11393
5.0	ITU-T Z.106	2016-04-29	17	11.1002/1000/12860
6.0	ITU-T Z.106	2019-10-14	17	11.1002/1000/14057

Keywords

Common Interchange Format, directives, SDL-2010, SDL-CIF, SDL-PR, Specification and Description Language, textual phrase representation.

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

FOREWORD

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

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

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

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

NOTE

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

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

INTELLECTUAL PROPERTY RIGHTS

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

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

Table of Contents

Introduction

For a number of years, the Specification and Description Language has been increasingly used, both in industry and for standards and Recommendations. Whereas, in industry, the Specification and Description Language is often used in an environment with a single tool, environments for standards and Recommendation creation often require the integration of Specification and Description Language specifications from many tools used by different organizations. This is often also a requirement in international projects.

Until the time Recommendation ITU-T Z.106 had been agreed, the only way to interchange specifications in the Specification and Description Language had been to interchange SDL-PR (the textual representation of the language). This has led to the loss of graphical information. Though not necessary from the formal viewpoint, graphical information often had a significant impact on readability and comprehensibility. With the common interchange format, this Recommendation fulfilled a long-expressed need for the interchange of specifications of the Specification and Description Language without the loss of graphical information.

Recommendation ITU-T Z.106

Specification and Description Language – Common interchange format for SDL-2010

1 Scope

This Recommendation defines the common interchange format (SDL-CIF) for specifications written in Specification and Description Language. It is intended for tool vendors as an export and import format to allow the interchange of specifications with tools offered by other tool vendors. Even though the format allows writing specifications in SDL-CIF directly, it is not intended for this purpose, but rather should be generated from an existing specification written according to the Specification and Description Language as defined in [ITU-T Z.100], [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104], [ITU-T Z.105] and [ITU-T Z.107].

2 References

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

[ITU-T Z.100] Recommendation ITU-T Z.100 (2019), *Specification and Description Language – Overview of SDL-2010*.

[ITU-T Z.101] Recommendation ITU-T Z.101 (2019), *Specification and Description Language – Basic SDL-2010*.

[ITU-T Z.102] Recommendation ITU-T Z.102 (2019), *Specification and Description Language – Comprehensive SDL-2010*.

[ITU-T Z.103] Recommendation ITU-T Z.103 (2019), *Specification and Description Language – Shorthand notation and annotation in SDL-2010*.

[ITU-T Z.104] Recommendation ITU-T Z.104 (2019), *Specification and Description Language – Data and action language in SDL-2010*.

[ITU-T Z.105] Recommendation ITU-T Z.105 (2019), *Specification and Description Language – SDL-2010 combined with ASN.1 modules*.

[ITU-T Z.107] Recommendation ITU-T Z.107 (2019), *Specification and Description Language – Object-oriented data in SDL-2010*.

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

3 Definitions and abbreviations

3.1 Definitions

None.

3.2 Abbreviations and acronyms

This Recommendation uses the following abbreviation and acronyms:

CIF-GR Common Interchange Format for Specification and Description Language Graphic Representation

CIF-PR Common Interchange Format for Specification and Description Language textual Phrase Representation

SDL-CIF Specification and Description Language Common Interchange Format

SDL-GR Specification and Description Language Graphic Representation

SDL-PR Specification and Description Language textual Phrase Representation

4 Conventions

For the definition of properties and syntaxes of SDL-CIF, the meta-language defined in clause 5.4.2 of [ITU-T Z.111] has been used with the following addition: the last alphanumeric part of a rule name gives the rules category (A, B or C) followed by an identifying number in that category. Non-terminal symbols refer to non-terminals of [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104] or [ITU-T Z.107] or to non-terminals introduced in this Recommendation.

The following non-terminal rules are intentionally defined but not used because they are starting rules for syntax:

<sdl specification>,
<cif description a1>,
<cif level 1 file>.

The following non-terminal rule is intentionally not used, because it is a general descriptive rule for SDL-CIF comments:

<cif directive>.

5 Level 0 SDL-CIF (SDL-PR)

5.1 General principles

SDL-CIF level 0 introduces an additional syntactic form for representing a system: the textual phrase representation (SDL-PR). It is allowed to use SDL-PR instead of the graphic representation of [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104] or [ITU-T Z.107], which is referred to as SDL-GR in this document. As both are concrete representations of the same abstract grammar, they are equivalent. In particular, they are both equivalent to the abstract grammar for the corresponding concepts.

A subset of SDL-GR is common with SDL-PR. This subset is called common textual grammar.

Although SDL-2010 is allowed to be written in either SDL-PR or SDL-GR, the language has been designed with the knowledge that SDL-PR will be used infrequently for purposes such as interchanging between tools, albeit the common interchange format specified in the remainder of this document further diminishes the use of SDL-PR. Most users use SDL-GR.

Figure 1 shows the relationships between SDL-PR, SDL-GR, the concrete grammars and the abstract grammar.

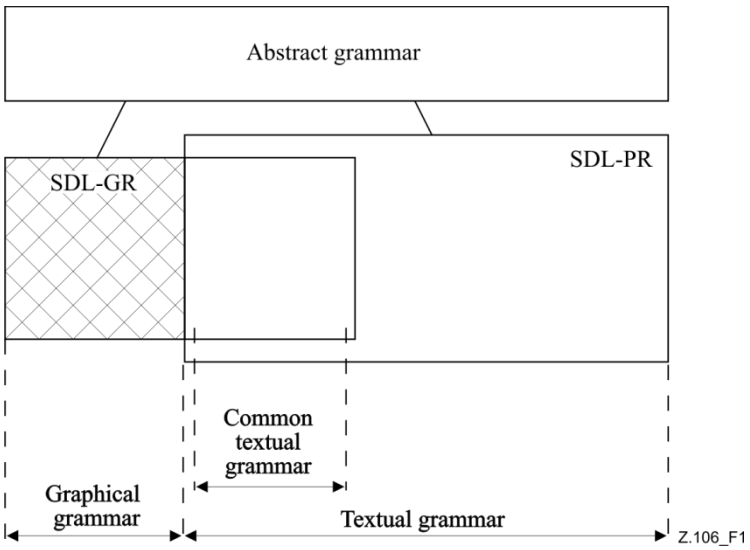


Figure 1 – SDL-2010 grammars

Each of the concrete grammars has a definition of its own syntax and of its relationship to the abstract grammar (that is, how to transform into the abstract syntax). Using this approach, there is only one definition of the semantics of SDL; each of the concrete grammars inherits the semantics via its relationship to the abstract grammar (in [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.104] and [ITU-T Z.107]). This approach also ensures that SDL-PR and SDL-GR are equivalent.

For some constructs there is no directly equivalent abstract syntax. In these cases, a model is given for the transformation from concrete syntax into the concrete syntax of other constructs that (directly or indirectly via further models) have an abstract syntax. Items that have no mapping to the abstract syntax (such as comments) do not have any formal meaning.

The concrete textual syntax for SDL-PR is applied as defined in *Concrete grammar* of clause 5.3.2 of [ITU-T Z.111] and is specified in the extended Backus-Naur Form of syntax description defined in clause 5.4.2 of [ITU-T Z.111]. Underlined semantic sub-categories (see clause 5.4.2 of [ITU-T Z.111]) are significant.

Along with the productions for the concrete syntax of SDL-PR, *Model* clauses are given for constructs that are "derived concrete syntax" for other equivalent concrete syntax constructs as defined in *Model* of clause 5.3.2 of [ITU-T Z.111].

NOTE – To aid understanding, the following titles of the clauses often correspond to the titles of related clauses in [ITU-T Z.101], [ITU-T Z.102], [ITU-T Z.103], [ITU-T Z.104] and [ITU-T Z.107].

5.2 General rules

In SDL-PR, the optional name or identifier in a definition after the ending keywords (**endsystem**, **endblock**, etc.) in definitions shall be syntactically the same as the name or identifier following the corresponding commencing keyword (**system**, **block**, etc., respectively).

5.3 Organization of SDL-2010 specifications

5.3.1 Framework

The starting production rule of clause 7.1 of [ITU-T Z.103] <sdl specification>, is replaced by the following production:

```
<sdl specification> ::=
                    {       { <package diagram> | <package definition> | <module definition>
                                    |       <system specification> | <textual system specification> }
                            <referenced definition>* }set
```

<module definition> ::=
ModuleDefinition

where the non-terminal **ModuleDefinition** is defined in ASN.1 as defined in clause 6 of [ITU-T Z.105].

<textual system specification> ::=
<agent definition>
| {<package use clause>}* <textual typebased agent definition>

The alternatives <package diagram> and <system specification> of <sdl specification> are not part of SDL-PR.

Model

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

A <textual system specification> 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 <package use clause> before a <textual typebased agent definition> of a <textual system specification> is derived syntax for a <package use clause> before the <system heading> in the <system definition> derived from the <textual typebased agent definition>.

5.3.2 Package

<package definition> ::=
{<package use clause>}*
<package heading> <end>
{<entity in package>}*
endpackage [<package name>] <end>

<entity in package> ::=
<agent type definition>
| <agent type reference>
| <package definition>
| <package reference>
| <signal definition list>
| <signal list definition>
| <remote variable definition>
| <data definition>
| <procedure definition>
| <procedure reference>
| <remote procedure definition>
| <composite state type definition>
| <composite state type reference>
| <select definition>
| <macro definition>

5.3.3 Referenced definition

The alternative <diagram> of <referenced definition> is not part of SDL-PR.

The production rule <definition> (see clause 7.3 of [ITU-T Z.104]), is replaced by the following production:

<definition> ::=
<package definition>
| <agent definition>
| <agent type definition>
| <composite state definition>
| <composite state type definition>
| <procedure definition>
| <operation definition>
| <macro definition>

Each <definition> item (except <macro definition> items) shall have a corresponding reference in the associated <package diagram> or <system specification>.

5.4 Structural concepts

5.4.1 Types, instances and gates

5.4.1.1 Structural type definitions

5.4.1.1.1 Agent type

<agent type definition> ::=
<system type definition> | <block type definition> | <process type definition>

5.4.1.1.2 System type

<system type definition> ::=
<package use clause>*
<system type heading> <end> <agent structure>
endsystem type [[<qualifier>] <system type name>] <end>

If there is a <system type name> after **endsystem type**, this shall be the same as the <system type name> in the <system type heading>. If there is a <qualifier> after **endsystem type**, this shall identify the qualifier of the identifier for the system type.

5.4.1.1.3 Block type

<block type definition> ::=
<package use clause>*
<block type heading> <end> <agent structure>
endblock type [[<qualifier>] <block type name>] <end>

If there is a <block type name> after **endblock type**, this shall be the same as the <block type name> in the <block type heading>. If there is a <qualifier> after **endblock type**, this shall identify the qualifier of the identifier for the block type.

5.4.1.1.4 Process type

<process type definition> ::=
<package use clause>*
<process type heading> <end> <agent structure>
endprocess type [[<qualifier>] <process type name>] <end>

If there is a <process type name> after **endprocess type**, this shall be the same as the <process type name> in the <process type heading>. If there is a <qualifier> after **endprocess type**, this shall identify the qualifier of the identifier for the process type.

5.4.1.2 Composite state type

<composite state type definition> ::=
<composite state type graph> | <state aggregation type>

<composite state type graph> ::=
{<package use clause>}*
<composite state type heading> <end>
<composite state structure>
e**ndsubstructure state type** [[<qualifier>] <composite state type name>] <end>

<state aggregation type> ::=
{<package use clause>}*
<state aggregation type heading> <end>
<aggregation structure>
e**ndsubstructure state type** [[<qualifier>] <composite state type name>] <end>

In <composite state type graph> and <state aggregation type>, if there is a <composite state type name> after **endsubstructure state type**, this shall be the same as the <composite state type name> in the <composite state type heading>. If there is a <qualifier> after **endsubstructure state type**, this shall identify the qualifier of the identifier for the composite state type.

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

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

The optional keyword **substructure** is added to <composite state type heading> and <state aggregation type heading> to match the keyword **endsubstructure** needed to close the <composite state type definition>.

5.4.1.2.1 Definitions based on types

<textual typebased agent definition> ::=
<textual typebased system definition>
| <textual typebased block definition>
| <textual typebased process definition>

The agent type denoted by <base type> in the type expression of a <textual typebased agent definition> shall contain an unlabelled start transition in its state machine.

5.4.1.2.2 System definition based on system type

<textual typebased system definition> ::=
<typebased system heading> <end>

5.4.1.2.3 Block definition based on block type

<textual typebased block definition> ::=
block <typebased block heading> <end>

5.4.1.2.4 Process definition based on process type

<textual typebased process definition> ::=
process <typebased process heading> <end>

5.4.1.2.5 Composite state definition based on composite state type

<textual typebased state partition def> ::=
state aggregation <typebased state partition heading> <end>

5.4.1.3 Gate

<gate in definition> ::=
<textual gate definition> | <textual interface gate definition>

<textual gate definition> ::=
gate <gate> [<encoding rules>] [**adding**] <gate constraint> <end>

<textual interface gate definition> ::=
gate { **in** | **out** }
with <interface identifier> [<encoding rules>] [<textual endpoint constraint>] <end>

It is only allowed to specify **adding** in a subtype definition and only for a gate defined in the supertype. When **adding** is specified for a <gate>, any <textual endpoint constraint>s and <signal list>s are additions to the <gate constraint>s of the gate in the supertype.

5.5 Agents

<agent definition> ::=
<system definition> | <block definition> | <process definition>

<agent structure> ::=
{ <entity in agent>}*
{ <interaction> | <agent body> }

<interaction> ::=
{ <channel to channel connection>
| <channel definition>
| <agent definition>
| <block reference>
| <process reference>
| <textual typebased agent definition> }*
[<state machine>]

The <state machine> defines the state machine of the agent. If <agent structure> is able to be understood both as either <state machine> or an <agent body>, it is interpreted as <agent body>.

A <channel to channel connection> shall not be contained within an <agent type definition>.

<entity in agent> ::=
<block type definition>
| <block type reference>
| <composite state definition>
| <composite state type definition>
| <composite state type reference>
| <data definition>
| <gate in definition>
| <imported procedure specification>
| <imported variable specification>
| <macro definition>
| <procedure definition>
| <procedure reference>
| <process type definition>
| <process type reference>
| <remote procedure definition>
| <remote variable definition>
| <select definition>
| <signal definition list>
| <signal list definition>
| <timer definition>
| <valid input signal set>
| <variable definition>

NOTE 1 – A composite state reference is not allowed as an <entity in agent>. The composite state is instead referenced by an item in a <state> of a <agent body> or a <state machine> of an <interaction>.

<block reference> ::=

block <block name> [**adding**] [<number of instances>] **referenced** <end>

<process reference> ::=

process <process name> [**adding**] [<number of instances>] **referenced** <end>

The <block reference> and <process reference> of [ITU-T Z.103] are extended to include the optional keyword **adding** for inherited agent definitions. If the keyword **adding** is present the agent reference shall be in a subtype and shall name an agent defined in the supertype. If **adding** is given, a <number of instances> item represents the *Number-of-instances* of the inherited *Agent-definition* in the subtype; otherwise the *Number-of-instances* is the same as the *Number-of-instances* of the inherited *Agent-definition* in the supertype.

<agent body> ::=

[<start>]
{ <state> | <free action> }*

<state machine> ::=

state substructure { <typebased composite state> | <composite state list item>}
referenced <end>

NOTE 2 – No notation is needed for inherited state machines, because any additional connection to the gates of the subtype state machine are given by a <channel definition> with a <channel path> where the <channel endpoint> is **this**.

5.5.1 System

<system definition> ::=

{<package use clause>}*
<system heading> <end> <agent structure>
endsystem [<system name>] <end>

5.5.2 Block

<block definition> ::=

{<package use clause>}*
<block heading> <end> <agent structure>
endblock [[<qualifier>] <block name>] <end>

If there is a <block name> after **endblock**, this shall be the same as the <block name> in the <block heading>. If there is a <qualifier> after **endblock**, this shall identify the qualifier of the identifier for the block.

5.5.3 Process

<process definition> ::=

{<package use clause>}*
<process heading> <end> <agent structure>
endprocess [[<qualifier>] <process name>] <end>

If there is a <process name> after **endprocess**, this shall be the same as the <process name> in the <process heading>. If there is a <qualifier> after **endprocess**, this shall identify the qualifier of the identifier for the process.

5.5.4 Procedure

<internal procedure definition> ::=

{<package use clause>}*
<procedure heading> [<end> <entity in procedure>+]
{ [<virtuality>] [<comment body>] <left curly bracket>
<statements> <end>*
<right curly bracket>
| [<procedure body>]
endprocedure [[<qualifier>] <procedure name>] <end> }

If there is a <procedure name> after **endprocedure**, this shall be the same as the <procedure name> in the <procedure heading>. If there is a <qualifier> after **endprocedure**, this shall identify the qualifier of the identifier for the procedure.

The syntax of <internal procedure definition> is extended to allow a <procedure body> as an alternative to <statements> enclosed in curly brackets.

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

An **endprocedure** within an inner <internal procedure definition> in an <entity in procedure> of an outer <internal procedure definition> belongs to the inner <internal procedure definition>.

<entity in procedure> ::=
<variable definition>
| <data definition>
| <select definition>
| <macro definition>
| <procedure definition>
| <procedure reference>
| <composite state definition>
| <composite state type definition>
| <composite state type reference>

The syntax of <entity in procedure> is extended to allow composite state and state type definitions to be used in a <procedure body>.

5.5.5 Agent and composite state reference

<composite state reference> ::=
state substructure <composite state name> **referenced** <end>

5.6 Communication

5.6.1 Channel

<channel definition> ::=
channel [<channel name> [<encoding rules>]] [**nodelay**]
<channel path> [<channel path>]
endchannel [<channel name>] <end>

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

<channel endpoint> ::=
{ <agent identifier> | <state identifier> | **env** | **this** }[**via** <gate>]

It is only allowed to specify the ending <channel name> if the starting <channel name> is specified. If the starting <channel name> is not specified, the channel shall not be referred to by name.

<gate> shall be specified if:

a) <channel endpoint> with an <agent identifier> denotes a connection to an agent definition, in which case the <gate> shall be defined directly in the agent type for that agent; or

b) <channel endpoint> with a <state identifier> identifies the state machine of the enclosing agent, in which case the <gate> shall be defined directly in the state type for that state, and as there is only one state machine it can alternatively be identified by **this**; or

c) **env** is specified and the channel is defined in an agent type in which case the <gate> shall be defined in this agent type.

If <gate> is specified, the channel is connected to that gate. The gate and the channel shall have at least one common element in their signal lists in the same direction. If no <gate> is specified, the following applies: If the channel endpoint is an agent or state machine and that agent/state contains

a <channel to channel connection> for the channel, the channel is connected to the implicit gate introduced by the <channel to channel connection>. Otherwise, if the channel endpoint is a state, the channel is connected to the implicit gate of that state machine; the channel introduces an implicit gate on the agent or state mentioned in <channel endpoint>. This gate obtains the <signal list> of the respective <channel path>s as its corresponding gate constraint. The channel is connected to that gate.

If a <channel definition> contains two <channel path>s, then the following conditions shall be satisfied.

a) The <channel endpoint> following **from** in the first <channel path> shall be the same as the <channel endpoint> following **to** in the second <channel path>.

b) The <channel endpoint> following **to** in the first <channel path> shall be the same as the <channel endpoint> following **from** in the second <channel path>.

NOTE – Clause 10.1 *Concrete grammar* of [ITU-T Z.103] contains a description of whether it is possible in the graphical grammar to derive the set of signals for a <signal list area> that has been omitted in a <channel definition area>. The equivalent set of rules for deriving an omitted <signal list> from a <channel path> of a <channel definition> is given below. Equivalent rules for the textual grammar are not normally given as they can usually be simply implied from the rules on the graphical grammar, but to make the rules for the textual grammar clear in this case an explicit description is given.

Derivation of an omitted channel <signal list> is possible if at least one <channel endpoint> identifies a <textual typebased agent definition> or <typebased composite state> (or **`this`** and the state machine is a <typebased composite state>) or **`env`**, and the gate for the <channel endpoint> has a defined set of signals in the direction for the <signal list>. The set of signals is defined for the gate if it identifies a <textual gate definition> that has a <gate constraint> with a defined <signal list>, or if a signal set is defined for all internal channels connected to this gate. The set is defined for a gate connected to <external channel identifiers> if for each external channel either no <signal list area> is omitted or the set for that external channel is derivable.

5.6.2 Connection

<channel to channel connection> ::=
connect <external channel identifiers>
and <channel identifier> {, <channel identifier>}* <end>

It is not allowed to mention a channel after the keyword **and** in more than one <channel to channel connection> of a given scope unit.

For any pair of <channel to channel connection>s of a given scope unit, the <external channel identifiers>s shall either mention the same set of channels, or shall have no channels in common. If two or more <channel to channel connection>s of a given scope unit have the same set of external channels, this is derived syntax for a single <channel to channel connection> having one of the <external channel identifiers> as its <external channel identifiers>, and <channel identifier> list formed by listing all the internal <channel identifier> items.

NOTE – Because of the **connect** construct with an (external) channel, there is a need to have a name in the corresponding textual version, whereas the channel is allowed to be anonymous in the graphical version of a specification. This is completely analogous to the case of <merge area>s in process or procedure graphs. A tool that converts the graphical version of a specification to a textual version should thus be able to generate implicit channel names.

Model

Each different <channel to channel connection> in a given scope unit defines one implicit gate on the scope unit. All channels in the <channel to channel connection> are connected to that gate in their respective scope units. The gate constraints of the implicit gate are derived from the channels connected to the gate.

The name of the gate is a unique and unambiguous derived name. In the surrounding scope unit, the <channel definition> that is identified by the <channel identifier> is extended with a **via** <gate> part. The **via** <gate> part is added to the <channel endpoint> that references the current scope unit and it mentions the implicit gate. Inside the scope unit, the channels that are associated with the external channel by means of the <channel to channel connection> are modified by extending the <channel endpoint> that mentions **env** with a **via** <gate> part for the implicit gate.

5.7 Behaviour

5.7.1 Start

<start> ::=

start [<virtuality>] [<state entry point name>] <end> <transition>

If <state entry point name> is given in a <start>, the <start> shall be the <start> of a <composite state definition>.

5.7.2 State

<state> ::=

state <state list> <end>
{ <input part>
| <priority input>
| <save part>
| <spontaneous transition>
| <continuous signal>
| <connect part> }*
[<state timer part>]
[**endstate** [<state name>] <end>]

<state timer part> ::=

input <virtuality> <state timer> <end> <transition>

It is allowed to specify the optional <state name> ending a <state> only if the <state list> in the <state> consists of a single <state name>, in which case it shall be that <state name>.

The <connect part> is only allowed for a <state> with <state list> that contains a <typebased composite state> or <composite state list item>.

5.7.3 Input

<input part> ::=

input [<virtuality>] <input list> <end>
[<enabling condition>] <transition>

5.7.4 Priority input

<priority input> ::=

priority input [<virtuality>]
<priority input list> <end> <transition>

5.7.5 Continuous signal

<continuous signal> ::=

provided [<virtuality>]
<continuous expression> <end>
[**priority** <priority name> <end>] <transition>

5.7.6 Enabling condition

<enabling condition> ::=

provided <provided expression> <end>

5.7.7 Save

<save part> ::=
save [<virtuality>] <save list> <end>

5.7.8 Spontaneous transition

<spontaneous transition> ::=
input [<virtuality>] <spontaneous designator> <end>
[<enabling condition>] <transition>

5.7.9 Label

<label> ::=
<connector name>:

NOTE – In the abstract grammar, only free actions have labels; labels inside of a transition are transformed into separate free actions.

<free action> ::=
connection
<transition>
[**endconnection** [<label>] <end>]

If the <transition string> of the <transition> in <free action> is non-empty, the first <action> shall have a <label>; otherwise the <terminator> shall have a <label>. If present, the <label> ending the <free action> shall be the same as this <label>.

Model

If a <label> is not the first label of a <transition string>, the <transition string> is split into two parts. All <action>s preceding the <label> are preserved in the original transition, which is terminated with a <join> to the <label>. All action statements following <label> are copied to a new <free action>, which starts with the <label>. If the <transition string> is followed by a <terminator> with a label, the new <free action> is terminated with a <join> to the label of the <terminator> and another <free action> is made that contains the labelled <terminator>. If the <transition string> is followed by a <terminator> without a label, the new <free action> is terminated with the unlabelled <terminator>. If a <transition string> without any labels is followed by a labelled <terminator>, a <free action> is made that contains the labelled <terminator> and the <terminator> is replaced by a <join> to the label of the <terminator>.

5.7.10 State machine and composite state

<composite state definition> ::=
<composite state graph> | <state aggregation>

5.7.10.1 Composite state graph

<composite state graph> ::=
{<package use clause>}*
<composite state heading> <end> <composite state structure>
endsubstructure [[<qualifier>] <composite state name>] <end>

<composite state heading> ::=
<virtuality> **state** [**substructure**] [<qualifier>] <state name>
[<specialization>] [<agent formal parameters>]

The optional keyword **substructure** is added to <composite state heading> to match the keyword **endsubstructure** needed to close the <composite state graph>.

If there is a <composite state name> after **endsubstructure**, this shall be the same as the <state name> in the <composite state heading>. If there is a <qualifier> after **endsubstructure**, this shall identify the qualifier of the identifier for the composite state.

<composite state structure> ::=
{ <state connection points> | <entity in composite state> }*
<composite state body>

<entity in composite state> ::=
<valid input signal set>
| <variable definition>
| <data definition>
| <select definition>
| <macro definition>
| <procedure definition>
| <procedure reference>
| <composite state definition>
| <composite state reference>
| <composite state type definition>
| <composite state type reference>
| <gate in definition>

An <entity in composite state> that is a <valid input signal set> is only valid if the corresponding *Composite-state-type-definition* is used for *State-machine* items of agent types. A <composite state reference> in an <aggregation structure> is never an <entity in composite state>, but is always a <state partition> and therefore ends the list of items (<state connection points> or <entity in composite state>) before the <state aggregation body>.

<composite state body> ::=
<start>* { <state> | <free action> }*

5.7.10.2 State aggregation

<state aggregation> ::=
{<package use clause>}*
<state aggregation heading> <end> <aggregation structure>
endsubstructure [[<qualifier>] <composite state name>] <end>

<state aggregation heading> ::=
<virtuality> **state aggregation** [**substructure**] [<qualifier>] <state name>
[<specialization>] [<agent formal parameters>]

The optional keyword **substructure** is added to <state aggregation heading> to match the keyword **endsubstructure** needed to close the <composite state graph>.

If there is a <composite state name> after **endsubstructure**, this shall be the same as the <state name> in the <state aggregation heading>. If there is a <qualifier> after **endsubstructure**, this shall identify the qualifier of the identifier for the composite state.

<aggregation structure> ::=
{ <state connection points> | <entity in state aggregation> }*
<state aggregation body>

<entity in state aggregation> ::=
<valid input signal set>
| <variable definition>
| <data definition>
| <select definition>
| <macro definition>
| <procedure definition>
| <procedure reference>
| <composite state type definition>
| <composite state type reference>
| <gate in definition>

NOTE 1 – An <entity in state aggregation> differs from an <entity in composite state> by excluding <composite state definition> and <composite state reference>, which in <aggregation structure> are <state partition> alternatives.

<state aggregation body> ::=
{ <state partition>
| <state partition connection> }*

<state partition> ::=
<textual typebased state partition def>
| <composite state reference>
| <composite state definition>

NOTE 2 – No notation is needed for inherited state partitions, because any additional connection to the gates of the subtype state partition is given by a <channel definition> with a <channel path> where the <channel endpoint> identifies the state partition.

<state partition connection> ::=
<state partition connection entry> | <state partition connection exit>

<state partition connection entry> ::=
connect [<composite state identifier> **via**] <outer entry points>
and <composite state identifier> **via** <inner entry point> <end>

<state partition connection exit> ::=
connect [<composite state identifier> **via**] <outer exit point>
and <composite state identifier> **via** <inner exit points> <end> }

In <state partition connection entry> and <state partition connection exit>, the optional <composite state identifier> after the keyword **connect** and the following keyword **via** has no SDL-2010 meaning. It is legacy syntax. However, if present the <composite state identifier> shall identify the state aggregation.

5.7.10.3 State connection point

<state connection points> ::=
{ **in** <state entry points> | **out** <state exit points> } <end>

5.7.10.4 Connect

<connect part> ::=
connect [<virtuality>] [<connect list>] <end>
<exit transition>

<exit transition> ::=
<transition>

5.7.11 Transition

5.7.11.1 Transition body

<transition> ::=
<transition action items>
| <terminator>

<transition action items> ::=
<transition string> [<terminator>]

<transition string> ::=
<action>+

<action> ::=
[<label>] <action item> <end>

<action item> ::=
<task>
| <output>
| <create request>
| <decision>
| <set>
| <reset>
| <export statement>
| <procedure call>
| <remote procedure call>
| <transition option>

<terminator> ::=
[<label>] <terminator node> <end>

<terminator node> ::=
<nextstate>
| <join>
| <stop>
| <return>

If the <terminator> of a <transition> is omitted, then the last action in the <transition> shall contain a terminating <decision> (see clause 5.7.12.5) or terminating <transition option>, except when a <transition> is contained in a <decision> or <transition option>.

NOTE – In the graphical grammar, <terminator area> has alternatives for <state area>, <merge area>, <decision area> and <transition option area>. A<state area> is transformed in the graphical grammar as described in clause 11.2 *Model* of [ITU-T Z.103] to a <nextstate area> as the <terminator area> of the <transition area> and a <state area> that is not a <terminator area> of a <transition area> (see *Model* in clause 11.12.1 of [ITU-T Z.103]). The equivalent textual grammar has a corresponding <terminator> that is a <nextstate> terminator and a <state>. A <merge area> is transformed in the graphical grammar as described in clause 11.12.2.2 *Model* of [ITU-T Z.103] to an <out connector area> to a unique <connector name> and attaching an <in connector area>, with the same <connector name> to the <flow line symbol> in the <merge area>. The equivalent textual grammar has <join> items for the <out connector area> items and a <label> for the <connector name>. The <decision area> and <transition option area> are topologically different to <decision> and <transition option> as a textual grammar <action item>, because in the textual grammar if a <terminator> is not specified the next item in the <transition string> follows, whereas in the graphical grammar each <transition area> of a <decision area> or <transition option area> has a <terminator>. Any <decision area> or <transition option area> as a <terminator area> in the graphical grammar, in the textual grammar becomes a <decision> or <transition option> (respectively) as the last <action item> of a <transition string> for the <transition string area> and the <terminator area>. In this case, the <transition string> is not followed by a <terminator>.

5.7.11.2 Transition terminator

5.7.11.2.1 Nextstate

<nextstate> ::=
nextstate <nextstate body>

5.7.11.2.2 Join

<join> ::=
join <connector name>

A <join> corresponds to an <out connector area> in SDL-GR.

There shall be exactly one <connector name> corresponding to a <join> within the same body.

5.7.11.2.3 Stop

<stop> ::=
stop

5.7.11.2.4 Return

<return> ::=
return { <return body> | **via** <state exit point> } [<end>]

5.7.12 Action

5.7.12.1 Task

<task> ::=
task { [<comment body>] <left curly bracket> <task body> <right curly bracket>
| <informal text>
| <legacy task body> } [<end>]
| <macro call>

NOTE – Macro calls are allowed to "appear at any place where a <lexical unit> is allowed" (clause 6.7.3 of [ITU-T Z.102]). For SDL-CIF level 0 (SDL-PR) <macro call> is expanded and removed before the syntax of <task> is considered, therefore <macro call> is redundant in <task> and does not apply for SDL-CIF level 0 (SDL-PR). It is included here for SDL-CIF level 1 (CIF-PR) and SDL-CIF level 2 (CIF-GR) to avoid the need to redefine <task>.

5.7.12.2 Create

<create request> ::=
create <create body>

5.7.12.3 Procedure call

<procedure call> ::=
call <procedure call body>

<remote procedure call> ::=
call <remote procedure call body>

5.7.12.4 Output

<output> ::=
output <output body>

5.7.12.5 Decision

<decision> ::=
decision <question> <end>
<textual decision body>
enddecision

<textual decision body> ::=
<textual answer part>+ [<textual else part>]

<textual answer part> ::=
([<answer>] **)** <colon> [<transition>]

<textual else part> ::=
else <colon> [<transition>]

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

The <answer> of <textual answer part> shall be omitted if and only if the <question> consists of the keyword **any**. In this case, a <textual else part> shall be absent.

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

5.7.13 Timer

<reset> ::=
reset <reset body>

<set> ::=
set <set body>

A <reset> represents a *Reset-node*; a <set> represents a *Set-node*.

5.8 Data

5.8.1 Data definitions

5.8.1.1 Data type definition

<data type definition> ::=
{<package use clause>}*
<type preamble> <data type heading> [<data type specialization>]
{ [<comment body>] <left curly bracket> <data type definition body>
<right curly bracket>
| <end> [<data type definition body> <data type closing> <end>]
}

The syntax of <data type definition> is extended to allow definition using <data type closing> as an alternative to curly brackets.

<data type closing> ::=
endvalue type [<data type name>]

If present in a <data type closing> the <data type name> shall be matched by the name in the corresponding <data type heading>.

5.8.1.2 Interface definition

<interface definition> ::=
{<package use clause>}*
[<virtuality>] <interface heading>
[<interface specialization>]
{ <end> [<entity in interface>* <interface closing>]
| [<comment body>]
<left curly bracket>
<entity in interface>*
<right curly bracket>
}
| <signal list definition>

The syntax of <interface definition> is extended to allow definition using <interface closing> as an alternative to curly brackets.

<interface closing> ::=
endinterface [<interface name>] <end>

If present in a <interface closing> the <interface name> shall be matched by the name in the corresponding <interface heading>.

5.8.1.3 Behaviour of operations

```
<operation definition> ::=
                        {<package use clause>}*
                        <operation heading>
                              [ <end> <entity in operation>+ ]
                        {     [ <comment body> ] <left curly bracket>
                                    <statements>
                               <right curly bracket>
                        |     <operation body>
                              { endoperator | endmethod } [ [<qualifier>] <operation name>] <end> }
```

If there is an <operation name> after **endoperator** or **endmethod**, this shall be the same as the <operation name> in the <operation heading>. If there is a <qualifier> after **endoperator** or **endmethod**, this shall identify the qualifier of the identifier for the operation.

The syntax of <operation definition> is extended to allow an <operation body> as an alternative to <statements> enclosed in curly brackets.

```
<operation body> ::=
                        <start> { <free action> }*
```

5.8.1.4 Syntypes

```
<syntype definition syntype> ::=
                  syntype <syntype name> <equals sign> <parent sort identifier>
                  {     [ <comment body> ] <left curly bracket>
                              [ { <default initialization> [ [<end>] <constraint> ] | <constraint> } <end> ]
                        <right curly bracket>
                  |     {     <default initialization> [ [<end>] <constraint> ] | <constraint> } <end>
                                    <syntype closing>
                        |     [<syntype closing>] }
                        <end>
                  }
```

The syntax of <syntype definition syntype> is extended to allow definition using <syntype closing> as an alternative to curly brackets.

```
<syntype definition data type> ::=
                  <type preamble> <data type heading> [<data type specialization>]
                  {     [ <comment body> ] <left curly bracket> <data type definition body> <constraint> <end>
                        <right curly bracket>
                  |     <end> [ <data type definition body> <constraint> <end> <data type closing> <end>]
                  }
```

The syntax of <syntype definition data type> is extended to allow definition using <syntype closing> as an alternative to curly brackets.

```
<syntype closing> ::=
                        endsyntype [<syntype name>]
```

If present in a <syntype closing> the <syntype name> shall be the same as the name as the <syntype name> at the start of the definition.

5.9 Generic system definition

5.9.1 Optional definition

The following replaces the <select definition> of [ITU-T Z.103].

```
<select definition> ::=
                        select if ( <Boolean simple expression> ) <end>
```

{ <signal definition list>
| <data definition>
| <variable definition>
| <timer definition>
| <macro definition>
| <remote variable definition>
| <remote procedure definition>
| <select definition>
| <operation definition>
| <block reference>
| <process reference>
| <agent type reference>
| <procedure definition>
| <imported variable specification>
| <imported procedure specification>
| <signal list definition>
| <agent type definition>
| <agent definition>
| <channel definition>
| <channel to channel connection>
| <composite state definition>
| <composite state reference>
| <composite state type definition>
| <composite state type reference>
| <package definition>
| <package reference>
| <procedure reference>
| <state machine>
| <state partition>}+

endselect <end>

5.9.2 Optional transition string

<transition option> ::=

alternative <alternative question> <end>
<textual decision body>
endalternative

6 Level 1 SDL-CIF (CIF-PR)

6.1 General principles

SDL-CIF level 1 is a relaxation of SDL-PR syntax: it does not bring any additional information about graphical presentation; it only enhances the suitability of SDL-PR as an interchange format.

It overcomes the main drawback of SDL-PR as an interchange format, which is that SDL-PR only describes syntactically complete SDL-2010 descriptions, while there is a need to interchange descriptions which are partial or not yet achieved. However, these descriptions shall be syntactically correct to be interchanged.

CIF-PR reuses large parts of the SDL-PR syntax. The production rules of SDL-PR that are reused in CIF-PR are not described again in this Recommendation (either for CIF-PR or CIF-GR) but are referenced by name.

The ITU-T Z.100 rules to transform SDL-PR into the abstract grammar also apply to parts of CIF-PR that are shared with SDL-PR, as far as it is possible according to the incompleteness of the CIF-PR descriptions.

6.2 CIF-PR syntax

6.2.1 SDL-CIF file

The starting production rule of SDL-PR, <sdl specification> (see clause 5.3.1), is replaced by the following production:

```
<cif level 1 file> ::=
                    {       <textual system specification>
                            | <textual definition>
                    } *
```

6.2.2 Macro call

SDL-2010 macro calls are allowed to "appear at any place where a <lexical unit> is allowed" (clause 6.7.3 of [ITU-T Z.102]).

CIF-PR macro calls shall only appear at the place of a task.

The alternative of the rule <task> (see clause 5.7.12.1) for <macro call> applies for CIF-PR.

7 Level 2 SDL-CIF (CIF-GR)

7.1 General principles

SDL-CIF level 2 is an extension of SDL-CIF level 1 with so-called "SDL-CIF directives" that describe the main characteristics of the graphical representation of objects.

SDL-CIF directives are placed before the object they are associated with. One design principle used when defining this Recommendation has been that all SDL-PR constructs that contain information that the SDL-CIF converter has to extract should have an associated SDL-CIF comment placed before the SDL-PR construct. This makes it possible for an SDL-CIF reading tool to scan for the next SDL-CIF comment, extract necessary information from the following SDL-PR and then start looking for the next SDL-CIF comment.

It is allowed to associate several SDL-CIF directives with the same object.

The first SDL-CIF directive for an object usually describes the layout of the main part of the object, while the following SDL-CIF directives for the same object describe the layout of subparts of the object.

SDL-CIF level 2 does not describe all the details of the graphical representation, as this would restrict too much the number of tools able to support SDL-CIF: some editors favour manual (user) layout of symbols while others favour automatic layout. Handling both manual and automatic layout is a complex problem that is difficult to solve when developing an the Specification and Description Language tool.

In order to face this issue, SDL-CIF directives are classified in three categories: mandatory, optional and tool-specific directives.

Mandatory directives describe graphical characteristics which are not able to be automatically computed, or whose automatic computation would almost certainly be too far from the user expectations, e.g., the layout of symbols and lines in interconnection diagrams.

These graphical characteristics are generally user-controlled in the main the Specification and Description Language editing tools.

Optional directives describe graphical characteristics that are able to be automatically computed, for instance, text layout inside symbols. For any optional information that is not given, tools should automatically compute a layout.

Tool-specific directives describe characteristics (graphical or not) that are not covered by mandatory or optional directives. They allow tool manufacturers to add new SDL-CIF directives in the storage format that will be analysed by their own tools only.

7.2 General principles of graphical information

7.2.1 The coordinate system

The unit used is 1/10 mm. The origin is the upper left corner of a page. The positive x-axis is to the right of the origin. The positive y-axis is below the origin.

7.2.2 Pages

It is allowed to split diagrams into pages, as described in [ITU-T Z.101] and [ITU-T Z.103].

However, SDL-CIF files are not structured according to pages, but according to the SDL-PR syntax. Pages are described by SDL-CIF comments inserted between some syntax units. It is allowed that each page consists of information from several non-adjacent syntax units.

Pages are independent drawing areas. Every coordinate shall be interpreted according to the current page name.

7.2.3 Classification for information

SDL-CIF graphical information is classified into 4 classes:

– information about symbols which look like graphical lines, usually just called "lines", that is, channels and flow lines in transitions;

– information about symbols which do not look like lines, usually just called "symbols", e.g., process symbols, output symbols;

– information about text;

– other information, e.g., page-splitting information.

7.2.4 Symbol representation

All information about symbol positions and sizes is mandatory information.

The position of a symbol is usually given by the coordinates of the upper left corner of its bounding box. There are a few exceptions, where the upper right corner is used instead (for reversed text extension symbols and reversed comment symbols).

The size of a symbol is given by the width and height of its bounding box.

Symbol-specific information is sometimes added. One example is that for symbols that exist in both a left and right version, a "Left" or "Right" keyword is (optionally) present.

7.2.5 Text representation

Text positions and sizes are optional information. They refer to the text bounding box and not the text itself.

7.2.6 About optional text positions

The specification of different kinds of text positions is optional in SDL-CIF. This means that a tool does not have to specify a text position when writing an SDL-CIF file. It does also mean that a tool reading an SDL-CIF file with a specified text position does not have to use that text position. Here are some guidelines:

When reading an SDL-CIF file, a tool should try to use the text positions found in the SDL-CIF file. If this is not possible, autolayout should be used instead.

If a text position is not given in the SDL-CIF file, autolayout should be used.

Some text positions are more important than others to retain the original SDL-GR layout of a diagram in SDL-CIF. A maker of a tool should concentrate on implementing support for these text positions first. Below, text positions are listed in groups. Text positions that are most important to preserve are in the first group.

– Group 1: channel name, signal list, gate name, select.

– Group 2: connect, answer flow line, gate reference, return.

– Group 3: diagram kernel heading, page name, system symbol, block symbol, process symbol, procedure symbol, operator symbol, state, save, task, set, reset, create, procedure call, procedure start, decision, continuous signal, enabling condition, transition option, join, label, macro call, macro outlet, input, priority input, output, text, package reference.

7.2.7 Line representation

The layout of lines is given by a list of coordinates: one for the start point of the line, one for every break point on the line, and one for the end point of the line.

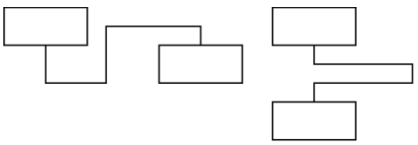
The start and end points of a line are usually connected to other symbols.

If a start or end point is connected to a symbol that does not have the shape of a rectangle, the point is on the symbol's bounding box instead of on the real symbol outline, to simplify geometry computations.

The layout of channel lines is mandatory information, as it is impossible to guess what the user wants to see.

7.2.8 About optional flow lines

Flow lines not following directly after a decision symbol are optional in the same sense as text positions. Some guidelines:



It is most important to give information about complicated flow lines like the two flowlines above.

It is less important to give information about simple flow lines like the two flowlines above.

Z.106_F2

Figure 2 – Flow line examples

7.2.9 Graphical information not covered by SDL-CIF

SDL-CIF directives are mainly related to graphical positions and sizes, because it is universal information that is interchangeable without implementation problems, and because it is information that would make diagrams very difficult to redraw if it was missing.

Some other kinds of information have been agreed as less important, and are not covered by SDL-CIF. These are information about text font, size of font, colour and line thickness.

It is allowed to include such information in tool-specific directives.

7.2.10 About nested diagrams

Nested (diagrams within diagrams) is not supported by SDL-2010. Nested diagrams should be converted so that a diagram within a diagram becomes a reference symbol and a separate diagram:

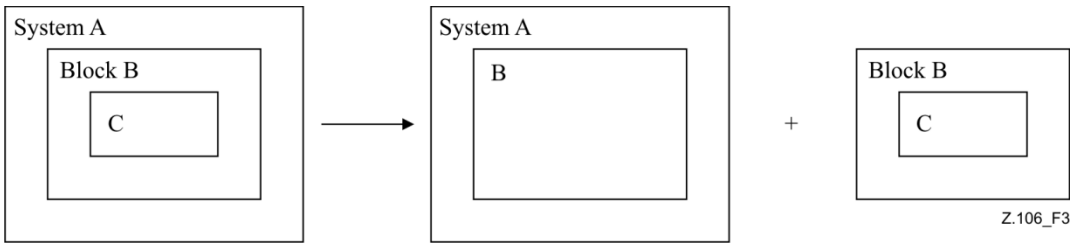


Figure 3 – Nested diagrams

7.2.11 About kernel and additional heading

SDL-CIF does not support additional headings. This means that tools supporting additional headings have to make the partitioning of the heading text into a kernel heading and an additional heading without support from SDL-CIF. Note that the SDL-GR in the example below is not correct, because according to Recommendation [ITU-T Z.100] there should not be a symbol around the additional heading text.

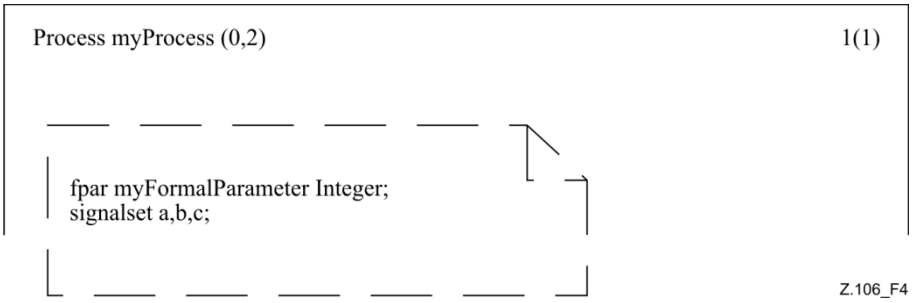


Figure 4 – A heading text split into kernel heading and additional heading

7.3 CIF-GR lexical rules

7.3.1 SDL-CIF directives

SDL-CIF directives are special forms of the ITU-T Z.100 <note> comments, all of which have in common the following description:

<cif directive> ::=
/* { **CIF** | **cif** } <text> */

A source line in the analysed file, which contains an SDL-CIF directive, shall not contain any other token.

A CIF-GR <note> shall not be a <cif directive>.

7.3.2 Newline and space characters

Newline and space characters are usually considered as non-significant characters when encountered during the analysis of the SDL-CIF file, and are then ignored.

However, when two adjacent SDL-2010 tokens are displayed in a diagram as adjacent parts of a text object, the newline and space characters between the two tokens should not be ignored: they should be used as a part of the text object.

This allows tools to keep the user preference for the text layout.

When two SDL-2010 tokens of a text object are separated by some space characters and a newline followed by several space characters, the space characters before the first significant character of the second line shall be ignored, as they are indentation spaces.

The first significant space character of a line is the character which is at the same column as the first '/' character of the previous SDL-CIF directive.

For example, in the following SDL-CIF fragment:

```
BLOCK b;
     /* CIF Channel (500,400),(300,400) */
     CHANNEL r FROM ENV To P WITH s1  ,   s2
          s3;
ENDCHANNEL;
```

The text which shall be displayed for the list of signals is:

```
's1  ,   s2' // NL // '  s3'
```

(provided there are no space characters after "s2").

7.3.3 About text layout

The placement of newline characters in SDL-GR should be preserved in SDL-PR, i.e., a signal list in SDL-GR with two signals on two lines should have the SDL-PR:

```
FROM ENV TO P WITH KeyStroke,
Card;
```

instead of:

```
FROM ENV TO P WITH KeyStroke, Card;
```

7.4 CIF-GR syntax: SDL-CIF A rules

The SDL-CIF A rules describe the SDL-CIF directives that are associated with different SDL-PR constructs. Usually there is one rule for each SDL-PR construct that corresponds to a graphical symbol. However, in some cases one symbol is described by more than one rule due to the fact that the information in SDL-PR is not collected together in one place. An example is the diagram frame where the start and end of a diagram are described by separate rules.

The SDL-CIF B rules are utility rules referenced from the SDL-CIF A rules and do not directly correspond to symbols in diagrams.

A normal SDL-CIF A rule is in general described like this:

– A section describing the grammar for the SDL-CIF comment and for related SDL-PR information. This section also shows how SDL-CIF comments should be placed in the SDL-PR specification.

– A section with explaining text and an example.

Note that this grammar only gives instructions on how to insert SDL-CIF comments into SDL-PR. A SDL-CIF level 2 file is correct if it both satisfies the SDL-CIF level 2 grammar (for the definition of SDL-CIF comments) and the SDL-CIF level 1 grammar.

7.4.1 A1 SDL-CIF description

<cif description a1> ::=
{ <diagram description a2> }*

Additional information

This is a rule that defines the structure of a valid SDL-CIF level 2 file. Note that this rule does not correspond to any symbol on a diagram.

7.4.2 A2 Diagram description

<diagram description a2> ::=
<diagram start a3> { <cif descriptor a19> | <page switch a21> }* <diagram end a18>

Additional information

This rule, together with the various rules for diagram start, <page switch a21> and the <diagram end a18> rule, specifies the frames and pages of diagrams.

7.4.3 A3 diagram start

<diagram start a3> ::=
<specification area start a4> | <package diagram start a5> | <system diagram start a6> | <system type diagram start a7> | <block diagram start a8> | <block type diagram start a9> | <process diagram start a10> | <process type diagram start a11> | <state diagram start a12> | <state type diagram start a13> | <state aggregation diagram start a14> | <state aggregation type diagram start a15> | <procedure diagram start a16> | <operator diagram start a17>

7.4.4 A4 specification area start

<specification area start a4> ::=
/* CIF SpecificationArea */
{ <page declaration b2> }+

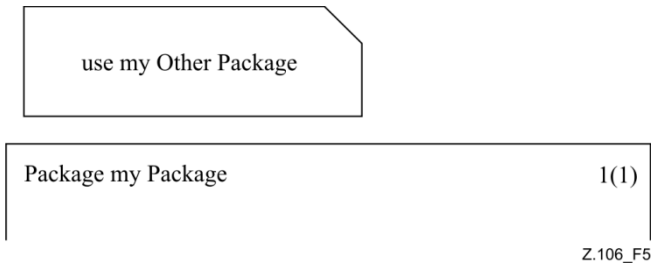
Additional information

There should be one <page declaration b2> for each page in the diagram.

7.4.5 A5 package diagram start

<package diagram start a5> ::=
/* CIF PackageDiagram */
<diagram parts b1>
{<package use clause>}* <package heading> <end>

Example




```
/* CIF PackageDiagram */
/* CIF Page 1 (1900,2300) */
/* CIF Frame (100,250),(1700,1950) */
/* CIF PackageReference (125,25) */

use myOtherPackage;

Package myPackage;
```

Figure 5

7.4.6 A6 system diagram start

<system diagram start a6> ::=
/* CIF SystemDiagram */
{ <diagram parts b1> }+
{ <package use clause> }*
<system heading> <end>

7.4.7 A7 system type diagram start

<system type diagram start a7> ::=
/* CIF SystemTypeDiagram */
<diagram parts b1>
<package use clause>* <system type heading> <end>

7.4.8 A8 block diagram start

<block diagram start a8> ::=
/* CIF BlockDiagram */
<diagram parts b1>
{<package use clause>}* <block heading>

7.4.9 A9 block type diagram start

<block type diagram start a9> ::=
/* CIF BlockTypeDiagram */
<diagram parts b1>
<package use clause>* <block type heading> <end>

7.4.10 A10 process diagram start

<process diagram start a10> ::=
/* CIF ProcessDiagram */
<diagram parts b1>
{<package use clause>}* <process heading> <end>

7.4.11 A11 process type diagram start

<process type diagram start a11> ::=
/* CIF ProcessTypeDiagram */
<diagram parts b1>
<package use clause>* <process type heading> <end>

7.4.12 A12 state diagram start

<state diagram start a12> ::=
/* CIF StateDiagram */
<diagram parts b1>
{<package use clause>}* <composite state heading> <end>

7.4.13 A13 state type diagram start

<state type diagram start a13> ::=
/* CIF StateTypeDiagram */:
<diagram parts b1>
{<package use clause>}* <composite state type heading> <end>

7.4.14 A14 state aggregation diagram start

<state aggregation diagram start a14> ::=
/* CIF StateAggregationDiagram */
<diagram parts b1>
{<package use clause>}* <state aggregation heading> <end>

7.4.15 A15 state aggregation type diagram start

<state aggregation type diagram start a15> ::=
/* CIF StateAggregationTypeDiagram */
<diagram parts b1>
{<package use clause>}* <state aggregation type heading> <end>

7.4.16 A16 procedure diagram start

<procedure diagram start a16> ::=
/* CIF ProcedureDiagram */
<diagram parts b1>
{<package use clause>}* <procedure heading> <end>

7.4.17 A17 operator diagram start

<operator diagram start a17> ::=
/* CIF OperatorDiagram */
<diagram parts b1>
{<package use clause>}* <operation heading> <end>

7.4.18 A18 diagram end

<diagram end a18> ::=
{ **/* CIF End SpecificationArea */** |
/* CIF End PackageDiagram */ ENDPACKAGE [<package name>] <end> |
/* CIF End SystemDiagram */ ENDSYSTEM [<system name>] <end> |
/* CIF End SystemTypeDiagram */ ENDSYSTEM TYPE [[<qualifier>] <system type name>] <end> |
/* CIF End BlockDiagram */ ENDBLOCK [[<qualifier>] <block name>] <end> |
/* CIF End BlockTypeDiagram */ ENDBLOCK TYPE [[<qualifier>] <block type name>] <end> |
/* CIF End ProcessDiagram ENDPROCESS [[<qualifier>] <process name>] <end> |
/* CIF End ProcessTypeDiagram */ ENDPROCESS TYPE [[<qualifier>] <process type name>] <end> |
/* CIF End StateDiagram */ ENDSUBSTRUCTURE [[<qualifier>] <composite state name>] <end> |
/* CIF End StateTypeDiagram */
ENDSUBSTRUCTURE STATE TYPE [[<qualifier>] <composite state type name>] <end> |
/* CIF End StateAggregationDiagram */ ENDSUBSTRUCTURE [[<qualifier>] <composite state name>] <end> |
/* CIF End StateAggregationTypeDiagram */
ENDSUBSTRUCTURE STATE TYPE [[<qualifier>] <composite state type name>] <end> |
/* CIF End ProcedureDiagram */ ENDPROCEDURE [[<qualifier>] <procedure name>] <end> |
/* CIF End OperatorDiagram */ ENDOPERATOR [[<qualifier>] <operator name>] <end>}

Example

```
/* CIF End SystemTypeDiagram */
ENDSYSTEM TYPE mySystemType;
```

7.4.19 A19 SDL-CIF descriptor

<cif descriptor a19> ::=
<diagram description a2> | <default size a20> | <channel a22> | <gate a23> | <gate symbol reference a24> |
<connect a25> | <state connection point a26> | <state connection a27> | <text extension a28> |
<comment a29> | <create line a30> | <flow line a31> | <answer flow line a32> | <block symbol a33> |
<dashed block symbol a34> | <process symbol a35> | <dashed process symbol a36> |
<package symbol a38> | <start symbol a41> |
<stop symbol a42> | <state symbol a43> | <nextstate symbol a44> | <save symbol a48> |
<task symbol a49> | <set symbol a50> | <reset symbol a51> | <export symbol a52> |
<create symbol a53> | <procedure call symbol a54> | <procedure start symbol a55> |
<return symbol a56> | <decision symbol a58> | <continuous signal symbol a59> |
<enabling condition symbol a60> | <transition option symbol a61> | <join symbol a62> |
<connect symbol a63> | <label symbol a64> | <input symbol a65> | <priority input symbol a66> |
<output symbol a67> | <text symbol a68> | <select symbol a69> | <descriptor end a70> |
<type reference a71> | <extended task symbol a75>

7.4.20 A20 default size

<default size a20> ::=
/* CIF DefaultSize <point b22> ***/**

Additional information

This comment is (optionally) placed before any symbol or line SDL-PR construct within a diagram.

The size given here will be used for all subsequent symbols without a defined size until a new default size is given. The default size is remembered even after a new <diagram start a3>. It is illegal to omit a symbol size specification before a default size is given.

Example where the task symbol will get the size (200,100):

```
/* CIF DefaultSize (200,100) */
/* CIF Task (800,550) */
task GameP:=null;
```

7.4.21 A21 page switch

<page switch a21> ::=
/* CIF CurrentPage <page name> ***/**

Additional information

This comment is (optionally) placed before any symbol or line SDL-CIF and SDL-PR construct within a diagram. The mentioned page becomes the current page. The page name shall refer to a page declared in the previous diagram start SDL-CIF comment.

Everything after this SDL-CIF comment in this diagram will be placed on the current page, until another current page is defined. The current page for a diagram is initially set by a <page declaration b2>.

Example

```
/* CIF CurrentPage 1 */
/* CIF Task (800,550) */
task GameP:=null;
```

7.4.22 A22 channel

<channel a22> ::=
/* CIF Channel <pointlist b18> [InvisibleName] ***/**
[<channel name text position b21>]
[<first signallist text position b4>]
[<second signallist text position b5>]
[<first arrow position b6>]
[<second arrow position b7>]
<channel definition>

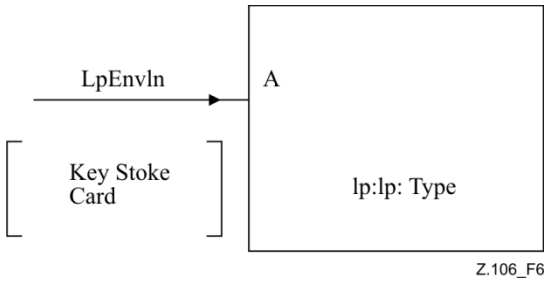
Additional information

If **InvisibleName** is given, the channel name should not appear in SDL-GR. The name is only given in SDL-PR to be able to refer to it in a **CONNECT** statement.

The first point in the pointlist is on the surrounding rectangle of the symbol corresponding to the **FROM** <channel endpoint> of the first <channel path> or on a gate connected to this symbol. The last point in the pointlist is on the surrounding rectangle of the symbol corresponding to the **TO** <channel endpoint> of the first <channel path> or on a gate connected to this symbol.

The text position of the gate in the **VIA** construct is specified in either <gate a23> or <gate reference b15>. The diagram below shows a gate and a reference to the gate.

Example 1




```
/* CIF Block (500,350) */
/* CIF GateReference (500,400) */
/* CIF TextPosition (510,390) */
BLOCK lp:lpType;
/* CIF Channel (300,400),(500,400) */
/* CIF TextPosition (390,350) */
/* CIF TextPosition (390,410) SignalList1 */
CHANNEL LpEnvIn
FROM env TO Lp VIA A
WITH KeyStroke,
Card;
ENDCHANNEL LpEnvIn;
```

Figure 6

7.4.23 A23 gate

<gate a23> ::=
/* CIF Gate <pointlist b18> [Dashed] ***/**
[<gate name text position b21>]
[<first signallist text position b4>]
[<second signallist text position b5>]
[<gate constraint symbol b3>]
<gate in definition>

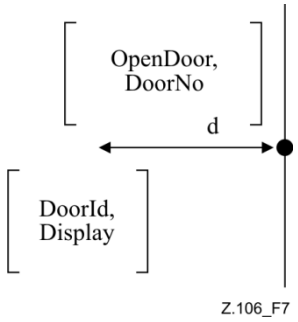
Additional information

There should be two points in the <pointlist b18>. The first point in the pointlist is on the surrounding rectangle of the frame symbol of the diagram. The second point in the pointlist should be the other point that defines the gate. If a <textual endpoint constraint> exists as part of <textual gate definition>, the second point will be on the surrounding rectangle of the symbol corresponding to the <textual endpoint constraint>.

Dashed should be used if the keyword **ADDING** is used in <textual gate definition>.

The <gate constraint symbol b3> should be given if a <textual endpoint constraint> is given.

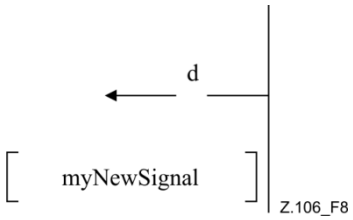
Example 1




```
/* CIF Gate (500,400),(300,400) */
/* CIF TextPosition (390,410) */
/* CIF TextPosition (490,410) SignalList1 */
/* CIF TextPosition (290,410) SignalList2 */
GATE d OUT
WITH DoorId,
Display;
IN
WITH OpenDoor,
DoorNo;
```

Figure 7

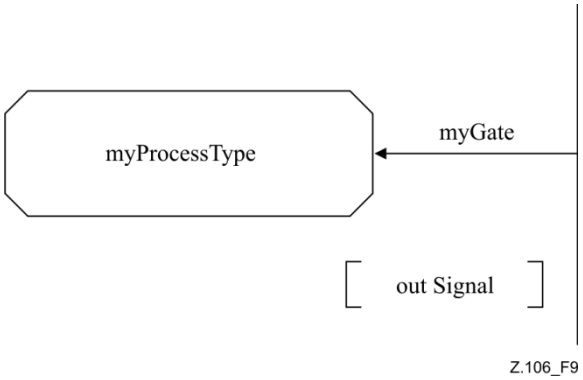
Example 2




```
/* CIF Gate (500,400),(300,400) Dashed */
GATE d ADDING OUT
WITH myNewSignal;
```

Figure 8

Example 3




```
/* CIF Gate (500,400),(300,400) */
/* CIF Process (100,350),(200,100) */
GATE myGate OUT TO myProcessType
WITH outSignal;
```

Figure 9

7.4.24 A24 gate symbol reference

<gate symbol reference a24> ::=
/* CIF GateSymbolReference <name> [**In** [<character string>]] [**Out** [<character string>]] <pointlist b18> ***/**
<text position b21>
[<in signal list position b39>]
[<out signal list position b40>]

Additional information

This SDL-CIF comment is used to specify gates on agent references and agent type references. Note that these gates have no corresponding SDL-PR.

The <text position b21> defines the position of the <name>.

If **In** is specified, then there is an arrow on the gate with direction towards the reference symbol. The <in signal list position b39> defines the position of the <character string> (describing the signal list) associated with **In**.

If **Out** is specified, then there is an arrow on the gate with direction from the reference symbol. The <out signal list position b40> defines the position of the <character string> (describing the signal list) associated with **Out**.

Example

```
     /* CIF GateSymbolReference g In 's1, s2' Out 's3, s4' ((800,550), (900,
550)) */
     /* CIF TextPosition (850,600) */
     /* CIF TextPosition (750,500) In */
     /* CIF TextPosition (850,600) Out */
```

7.4.25 A25 connect

<connect a25> ::=
/* CIF Connect [<position point b22>] ***/**
[<text position b21>]
<channel to channel connection>

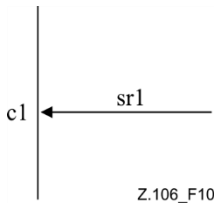
The production for <channel to channel connection> is defined in clause 5.6.2.

Additional information

<position point b22> is the position of the channel on the frame symbol.

<text position b21> is the text position for the <external channel identifiers>, i.e., the text outside the frame symbol.

Example




```
/* CIF Connect */
/* CIF TextPosition (800,50) */
CONNECT c1 AND sr1;
```

Figure 10

7.4.26 A26 state connection point

<state connection point a26> ::=
/* CIF StateConnectionPoint <pointlist b18> ***/**
[<text position b21>]
<state connection points>

Additional information

There should be two points in the <pointlist b18>. The first point in the pointlist shall be on the surrounding rectangle of the frame symbol of the diagram. The second point in the pointlist defines the other end of the state connection point symbol. If a <text position b21> exists, it specifies the position of the text in the <state connection point area>.

7.4.27 A27 state connection

<state connection a27> ::=
/* CIF StateConnection <pointlist b18> */
[<text position b21>]
<state partition connection>

The production for <state partition connection> is defined in clause 5.7.10.2.

Additional information

The first point in the pointlist is on the surrounding rectangle of the symbol corresponding to the <inner entry point>. The last point in the pointlist is on the surrounding rectangle of the frame symbol of the current page.

The <text position b21> specifies the position of the text in <inner entry point>, i.e., name(s) of the entry point(s) associated with the connected state symbol.

7.4.28 A28 text extension

<text extension a28> ::=
/* CIF TextExtension <position and size b20> [{ **Left** | **Right** }] ***/**
[<text position b21>] [<line b16>] <text>
/* CIF End TextExtension */

Additional information

Left means that the left side of the symbol is open. **Right** means that the right side of the symbol is open. **Right** is default.

If **Left** is given, the symbol and text position defines the upper right corner.

The <line b16> is the line connecting the text extension symbol with the other symbol. If the line is not given, it will be laid out automatically. The first point in the pointlist is on the surrounding rectangle of the text extension symbol. The last point in the pointlist is on the surrounding rectangle of the symbol the text extension symbol is attached to.

A newline character before or after one of the two SDL-CIF text extension comments should not be considered as a part of the text in the symbols.

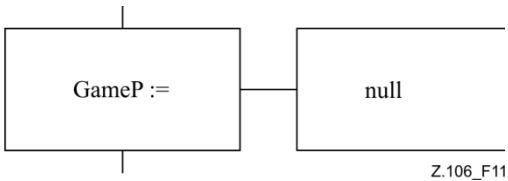
<text extension a28> and <comment a29> should be placed before the <end> in the following way: text extensions and comments are (as needed) attached to any rule in the range <block symbol a33> – <select symbol a69>.

When a text extension is attached to the task symbol that contains a <compound statement>, the text extension will be placed before the <right curly bracket> in the compound statement.

Example 1 (an example with an informal task symbol)

```
/* CIF Task (800,550) */
TASK 'first part of task text that will be in the task symbol'
/* CIF TextExtension (1100,550) */
'last part of task text that will be in the TextExtension symbol'
/* CIF End TextExtension */
;
```

Example 2




```
/* CIF Task (800,550) */
TASK GameP: =
/* CIF TextExtension (1100,550) */
/* CIF Line (1100,600),(1000,600) */
null
/*    CIF    End    TextExtension    */
;
```

Figure 11

7.4.29 A29 comment

<comment a29> ::=
/* CIF Comment <position and size b20> [**Left** | **Right**] [**Dashed**] ***/**
[<text position b21>] [<dashed line b17>]
<comment> <end>

Additional information

Left means that the left side of the symbol is open. **Right** means that the right side of the symbol is open. **Right** is default.

If **Left** is given, the symbol (and text) position defines the upper right corner.

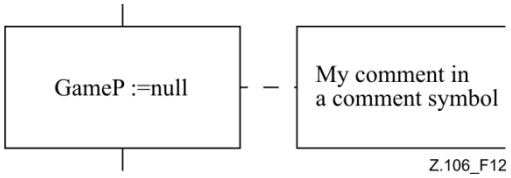
If **Dashed** is given, the comment symbol should be drawn dashed (as a <comment symbol>). If **Dashed** is not given, the comment symbol should be drawn non-dashed (as a <text extension symbol>).

The <dashed line b17> is the line connecting the comment symbol with the other symbol. If the line is not given, it will be laid out automatically. The first point in the pointlist is on the surrounding

rectangle of the comment symbol. The last point in the pointlist is on the surrounding rectangle of the symbol the comment is attached to.

How this SDL-CIF construct is used is explained in <text extension a28>.

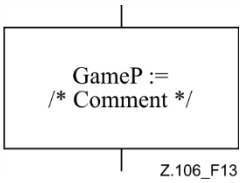
Example 1




```
/* CIF Task (800,550) */
task GameP:=null
/* CIF Comment (1100,550) */
/* CIF Line (1100,600),(1000,600) Dashed */
COMMENT 'My comment in
a comment symbol';
```

Figure 12

Related example




```
/* CIF Task (800,550) */
task GameP:=null
/* Comment */;
```

Figure 13

Example 2

```
/* CIF Task (800,550) */
task GameP: =
/* CIF TextExtension (1100,550) */
/* CIF Line (1100,600),(1000,600) */
null
/* CIF End TextExtension */
/* CIF Comment (1100,750) */
/* CIF Line (1100,800),(1000,600) Dashed */
COMMENT 'My comment in a comment symbol';
```

7.4.30 A30 create line

<create line a30> ::=
/* CIF CreateLine <pointlist b18> */

Additional information

The first point in the pointlist is on the surrounding rectangle of the process symbol that creates the other process. The last point in the pointlist is on the surrounding rectangle of the process symbol that is created.

Example

```
/* CIF DefaultSize (200,100) */
/* CIF Process (200,500) */
PROCESS Main(1,1) REFERENCED;
/* CIF Process (500,500) */
PROCESS Game(0,1) REFERENCED;
/* CIF CreateLine (400,550),(500,550) */
```

7.4.31 A31 flow line

<flow line a31> ::=
<line b16>

Additional information

This SDL-CIF comment is optional. If the SDL-CIF comment is not given for a flow line, the flow line is laid out automatically.

The first point in the pointlist is on the surrounding rectangle of the symbol the flow is coming from. The last point in the pointlist is on the surrounding rectangle of the symbol the flow is going to.

This comment is (optionally) placed before or after any symbol or line SDL-CIF and SDL-PR construct within the body of the diagram.

A flow line joining another flow line should describe the complete pointlist from a point on the surrounding rectangle of the "from" symbol to a point on the surrounding rectangle of the "to" symbol.

Arrows on flow lines are implicit. Flow lines after decision and transition option symbols should use <answer flow line a32>, an SDL-CIF rule that is not optional.

Example

```
/* CIF Start (300,100) */
START;
/* CIF Line (400,200),(400,250) */
/* CIF Set (300,250) */
Set(Now+1,T);
```

7.4.32 A32 answer flow line

<answer flow line a32> ::=
/* CIF Answer [{ **Right** | **Left** }] [**InvisibleBrackets**] */
[<line b16>] [<text position b21>]
{ <textual answer part> | <textual else part> }

This rule is used by <CIF descriptor a19>.

The productions for <textual answer part> and <textual else part> are defined in clause 5.7.12.5.

Additional information

This SDL-CIF comment should be used for flow lines after a decision or a transition option symbol.

If <line b16> is given:

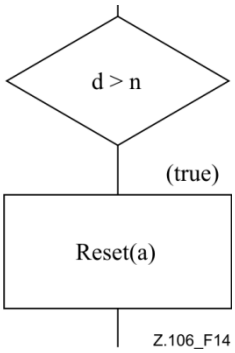
Right and **Left** have no meaning. The pointlist in the flow line specifies where the flow line starts on the decision symbol. The same rules as for <flow line a31> apply.

If <line b16> is not given:

Right means that the flow line starts to the right of the decision symbol (or in the lower right corner of the transition option symbol). **Left** means that the flow line starts to the left of the decision symbol (or in the lower left corner of the transition option symbol). As default, the flow line starts below the decision symbol (or in the centre of the bottom edge of the transition option symbol). The rest of the flow line is laid out automatically.

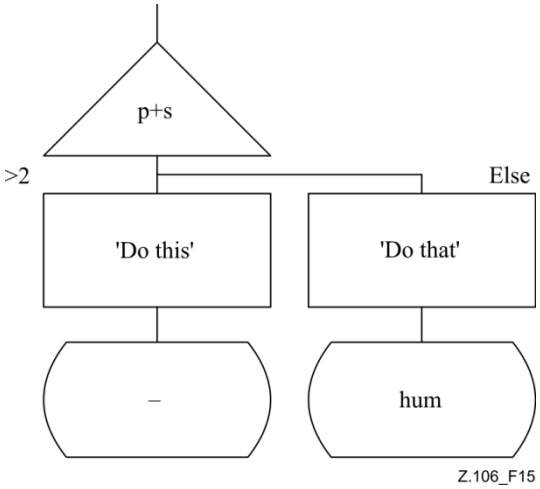
If **InvisibleBrackets** is given, the characters **(** and **)** found in the SDL-PR are not visible in SDL-GR.

Examples




```
/* CIF Decision (800,550) */
DECISION d > n;
/* CIF Answer */
/* CIF Line (900,650),(900,750) */
/* CIF TextPosition (910,690) */
(true):
/* CIF Reset (800,750) */
reset(a);
------
```

Figure 14




```
/* CIF Alternative (800,550) */
ALTERNATIVE p + s;
/* CIF Answer InvisibleBrackets */
(>2):
/* CIF Task (800,750) */
TASK 'Do this';
/* CIF NextState (800,950) */
NEXTSTATE -;
/* CIF Answer */
ELSE:
/* CIF Task (1100,750) */
TASK 'Do that';
/* CIF NextState (1100,950) */
NEXTSTATE hum
```

Figure 15

7.4.33 A33 block symbol

<block symbol a33> ::=
<block symbol rectangle b12> [<text position b21>]
{ <gate reference b15>* <block reference> | { <gate reference b15>* <textual typebased block definition> } }

This rule is used by <CIF descriptor a19>.

Additional information

<gate reference b15> items are optional and only used to specify text positions for gate references attached to this block symbol. If a <gate reference b15> is omitted, the text position for that gate reference will be laid out automatically. The name of the gate reference is in SDL-PR mentioned in connection with the SDL-PR for connected channels.

Example 1

```
/* CIF Block (800,550) */
/* CIF TextPosition (810,560) */
BLOCK myBlock REFERENCED;
```

Example 2

```
/* CIF Block (800,550) */
/* CIF GateReference (900,550) */
/* CIF TextPosition (890,500) */
BLOCK myBlocks(2):myBlockType;
```

7.4.34 A34 dashed block symbol

<dashed block symbol a34> ::=
/* CIF Block <position and size b20> **Dashed** <block name> ***/**
[<text position b21>] <gate reference b15>*

Additional information

This comment is (optionally) placed anywhere a textual <block reference> is allowed. There should be one <dashed block symbol a34> for each <inherited block definition> in the SDL-GR. <gate reference b15> is explained in <block symbol a33>.

Example

```
/* CIF Block (800,550) Dashed myBlock */
```

7.4.35 A35 process symbol

<process symbol a35> ::=
<process symbol rectangle b13>
[<text position b21>]
{ <gate reference b15>* <process reference> |
{ <gate reference b15>* <textual typebased process definition> } }

Additional information

<gate reference b15> is explained in <block symbol a33>.

Example 1

```
/* CIF Process (800,550) */
PROCESS myProcess REFERENCED;
```

Example 2

```
/* CIF Process (800,550) */
PROCESS myProcess (1,1):myProcessType;
```

7.4.36 A36 dashed process symbol

<dashed process symbol a36> ::=
/* CIF Process <position and size b20> **Dashed** <process name> ***/**
[<text position b21>] <gate reference b15>*

Additional information

This comment is (optionally) placed anywhere a textual <process reference> is allowed. There should be one <dashed process symbol a36> for each <inherited process definition> in the SDL-GR. <gate reference b15> is explained in <block symbol a33>.

Example

```
/* CIF Process (800,550) Dashed myProcess */
```

7.4.37 A38 package symbol

<package symbol a38> ::=
/* CIF Package <position and size b20> ***/**
[<text position b21>]
<package reference>

Additional information

This SDL-CIF comment shall only be used for package references in package diagrams, where there exists a SDL-PR syntax for package references. When describing package references in specification areas the rule 'package ref in specification area' shall be used.

Example

```
/* CIF Package (800,550) */
PACKAGE myPKG REFERENCED;
```

7.4.38 A41 start symbol

<start symbol a41> ::=
/* CIF Start <position and size b20> ***/**
[<text position b21>]
start [<virtuality>] [<state entry point name>] <end>

Additional information

This SDL-CIF comment should be used in agent and agent type diagrams. Procedure and operator diagrams should use <procedure start symbol a55>.

Example

```
/* CIF Start (800,550) */
START;
```

7.4.39 A42 stop symbol

<stop symbol a42> ::=
/* CIF Stop <position and size b20> ***/**
<stop> <end>

Example

```
/* CIF Stop (800,550) */
STOP;
```

7.4.40 A43 state symbol

<state symbol a43> ::=
/* CIF State <position and size b20> ***/**
[<text position b21>]
state <state list> <end>

Examples for this SDL-CIF comment are in the related SDL-CIF comment <nextstate symbol a44>.

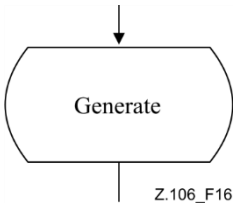
7.4.41 A44 nextstate symbol

<nextstate symbol a44> ::=
/* CIF NextState <position and size b20> ***/**
[<text position b21>]
<nextstate>

Additional information

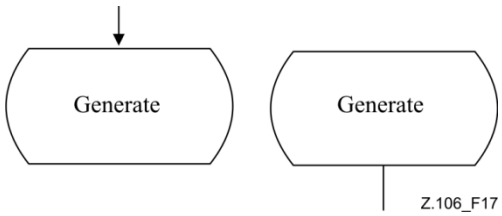
An SDL-CIF comment should be given for every state and every nextstate in SDL-PR. This means that there will be two SDL-CIF comments for one SDL-GR symbol that corresponds to both a SDL-PR state and a SDL-PR nextstate. A tool that reads an SDL-CIF file should determine if a state and a nextstate is in fact one SDL-GR symbol by comparing the coordinates of the symbols in the two SDL-CIF comments.

Examples




```
/* CIF NextState (800,550) */
NEXTSTATE Generate;
/* CIF State (800,550) */
STATE Generate;
------
```

Figure 16




```
/* CIF NextState (800,550) */
NEXTSTATE Generate;
/* CIF State (1100,550) */
STATE Generate;
```

Figure 17

7.4.42 A48 save symbol

<save symbol a48> ::=
/* CIF Save <position and size b20> ***/**
[<text position b21>]
<save part>

Example

```
/* CIF Save (800,550) */
SAVE mySignal;
```

7.4.43 A49 task symbol

<task symbol a49> ::=
/* CIF Task <position and size b20> ***/**
[<text position b21>]
<task> <end>

Additional information

There are three cases when a task symbol should be described by other SDL-CIF comments than this one. SDL-GR task symbols containing <set> should use <set symbol a50>. SDL-GR task symbols containing <reset> should use <reset symbol a51>. An SDL-GR task symbol containing a single <export statement> should use <export symbol a52>.

NOTE – <extended task symbol a75> is allowed as an alternative syntax.

Example

```
/* CIF Task (800,550) */
TASK myVariable: = 0;
```

7.4.44 A50 set symbol

<set symbol a50> ::=
/* CIF Set <position and size b20> ***/**
[<text position b21>]
<set> <end>

Additional information

A set symbol is a SDL-GR task symbol containing a single <set statement> (see clause 11.14 of [ITU-T Z.101]).

Example

```
/* CIF Set (800,550) */
SET (Now+1, myTime);
```

7.4.45 A51 reset symbol

<reset symbol a51> ::=
/* CIF Reset <position and size b20> ***/**
[<text position b21>]
<reset> <end>

Additional information

A reset symbol is a SDL-GR task symbol containing a single <reset statement> (see clause 11.14 of [ITU-T Z.101]).

Example

```
/* CIF Reset (800,550) */
RESET T;
```

7.4.46 A52 export symbol

<export symbol a52> ::=
/* CIF Export <position and size b20> ***/**
[<text position b21>]
<export statement> <end>

Additional information

An export symbol is a SDL-GR task symbol containing a <non-terminating statement> (see clause 11.13.1 of [ITU-T Z.101]) where the statement is a single <export statement> (see clause 11.14 of [ITU-T Z.102]).

Example

```
/* CIF Export (800,550) */
Export (myVariable1, myVariable2);
```

7.4.47 A53 create symbol

<create symbol a53> ::=
/* CIF Create <position and size b20> ***/**
[<text position b21>]
<create request> <end>

Example

```
/* CIF Create (800,550) */
CREATE Game;
```

7.4.48 A54 procedure call symbol

<procedure call symbol a54> ::=
/* CIF ProcedureCall <position and size b20> ***/**
[<text position b21>]
<procedure call> <end>

Example

```
/* CIF ProcedureCall (800,550) */
CALL myProcedure;
```

7.4.49 A55 procedure start symbol

<procedure start symbol a55> ::=
/* CIF ProcedureStart <position and size b20> ***/**
[<text position b21>]
start [<virtuality>] [<state entry point name>] <end>

Additional information

This SDL-CIF comment should be used for procedure and operator diagrams. Agent, agent type, state and state type diagrams should use <start symbol a41>.

Example

```
/* CIF ProcedureStart (800,550) */
START;
```

7.4.50 A56 return symbol

<return symbol a56> ::=
/* CIF Return <position and size b20> ***/**
[<text position b21>]
<return> <end>

Example

```
/* CIF Return (800,550) */
RETURN myReturnValue;
```

7.4.51 A58 decision symbol

<decision symbol a58> ::=
/* CIF Decision <position and size b20> ***/**
[<text position b21>]
decision <question> <end>

Additional information

A flow line directly after a decision symbol should be described by an <answer flow line a32>.

Example (without flow lines)

```
/* CIF Decision (800,550) */
DECISION DoorIndex > NoOfDoors;
```

7.4.52 A59 continuous signal symbol

<continuous signal symbol a59> ::=
/* CIF ContinuousSignal <position and size b20> ***/**
[<text position b21>]
provided [<virtuality>] <continuous expression> <end> [**priority** <priority name> <end>]

Example

```
/* CIF ContinuousSignal (800,550) */
PROVIDED level > 5;
```

7.4.53 A60 enabling condition symbol

<enabling condition symbol a60> ::=
/* CIF EnablingCondition <position and size b20> ***/**
[<text position b21>]
<enabling condition>

Example

```
/* CIF EnablingCondition (800,550) */
PROVIDED level > 5;
```

7.4.54 A61 transition option symbol

<transition option symbol a61> ::=
/* CIF TransitionOption <position and size b20> ***/**
[<text position b21>]

ALTERNATIVE <alternative question> <end>

Additional information

A flow line directly after a transition option symbol should be described by an <answer flow line a32>.

Example

```
/* CIF TransitionOption (800,550) */
ALTERNATIVE level;
```

7.4.55 A62 join symbol

<join symbol a62> ::=
/* CIF Join { <position and size b20> | **Invisible** } ***/**
[<text position b21>]
<join> <end>

Additional information

Invisible means that this SDL-PR join should not be visible as a symbol in SDL-GR; it is only given in SDL-PR to indicate a flow line ending in an already described symbol (i.e., the symbol following the label <join> is referring to). See also <label symbol a64>.

Example 1

```
/* CIF Join Invisible */
JOIN myInvisibleLabel;
```

Example 2

```
/* CIF Join (800,550) */
JOIN myLabel;
```

7.4.56 A63 connect

<connect symbol a63> ::=
/* CIF Connect */
[<line b16>] [<text position b21>]
connect [<virtuality>] [<connect list>] <end>

Additional information

This SDL-CIF comment should be used for flow lines from a state symbol to a transition without a starting input or continuous signal.

7.4.57 A64 label symbol

<label symbol a64> ::=
/* CIF Label { <position and size b20> | **Invisible** } ***/**
[<text position b21>]
{ <label> | **CONNECTION** <label> }

Additional information

Invisible means that this SDL-PR label should not be visible as a symbol in SDL-GR; it is only given in SDL-PR to indicate a flow line ending in an already described symbol (i.e., the symbol following the label). See also <join symbol a62>.

The first <label> in a <free action> has its SDL-CIF comment located before CONNECTION. See the third example below.

Example 1

```
/* CIF Label Invisible */
myInvisibleLabel:
```

Example 2

```
/* CIF Label (800,550),(100,100) */
myVisibleLabel:
```

Example 3

```
/* CIF Label (800,550), (100,100) */
CONNECTION myLabel:
```

7.4.58 A65 input symbol

<input symbol a65> ::=
/* CIF Input <position and size b20> [{ **Left** | **Right** }] ***/**
[<text position b21>]
{ **input** [<virtuality>] <input list> <end> | **input** [<virtuality>] <spontaneous designator> <end> }

Additional information

Left means that the part of the symbol that visualizes an arrow is to the left. **Right** means that the part of the symbol that visualizes an arrow is to the right. Default is **Right**.

Example

```
/* CIF Input (800,550) Left */
INPUT mySignal;
```

7.4.59 A66 priority input symbol

<priority input symbol a66> ::=
/* CIF PriorityInput <position and size b20>
[{ **Left** | **Right** }] ***/**
[<text position b21>]
priority input [<virtuality>] <priority input list> <end>

Additional information

Left means that the part of the symbol that visualizes an arrow is to the left. **Right** means that the part of the symbol that visualizes an arrow is to the right. Default is **Right**.

Example

```
/* CIF PriorityInput (800,550) Left */
PRIORITY INPUT mySignal;
```

7.4.60 A67 output symbol

<output symbol a67> ::=
/* CIF Output <position and size b20> [{ **Left** | **Right** }] ***/**
[<text position b21>]
<output> <end>

Additional information

Left means that the part of the symbol that visualizes an arrow is to the left. **Right** means that the part of the symbol that visualizes an arrow is to the right. Default is **Right**.

Example

```
/* CIF Output (800,550) */
OUTPUT mySignal;
```

7.4.61 A68 text symbol

<text symbol a68> ::=
/* CIF Text <position and size b20> ***/**
[<text position b21>]
<text>
/* CIF End Text */

Additional information

<text> shall not contain a line matching **'/* CIF End Text */'**

A newline character before or after one of the two SDL-CIF text comments should not be considered as a part of the text in the text symbol.

Example

```
/* CIF Text (800,550) */
Timer myTimer;
/* CIF End Text */
```

7.4.62 A69 select symbol

<select symbol a69> ::=
/* CIF Select <pointlist b18> ***/**
[<text position b21>]
SELECT IF <Boolean simple expression> <end>

Additional information

The points in the pointlist describe all the corners of the select symbol in either clockwise or counter clockwise order.

Example

```
/* CIF Select (700,400),(1100,400),(1100,750),(700,750) */
/* CIF TextPosition (725,425) */
SELECT IF (p = 3);
```

7.4.63 A70 descriptor end

<descriptor end a70> ::=
{ /* **CIF End Decision** */ **ENDDECISION** <end> |
/* **CIF End State** */ **ENDSTATE** [<state name>] <end> |
/* **CIF End Label** */ **ENDCONNECTION** [<connector name>] <end> |
/* **CIF End Select** */ **ENDSELECT** <end> |
/* **CIF End TransitionOption** */ **ENDALTERNATIVE** <end>
<end> **}**

Additional information

This rule is introduced to distinguish end of information about a symbol from end of information about a diagram.

7.4.64 A71 type reference

<type reference a71> ::=
/* CIF TypeReference <position and size b20> **Iconized */**
[<id text position b23>]
{{{**system** | **block** | **process** | **state**} **type** } | **procedure** } <identifier>

Additional information

Iconized is part of the SDL-CIF comment where a type reference symbol uses the special symbols for block/process/state types or procedures. The use of a class symbol is not supported for SDL-2010.

Example

```
/*CIF TypeReference (500,400) */
process type P referenced;
```

7.4.65 A75 extended task symbol

<extended task symbol a75> ::=
/* CIF Extendedtask <position and size b20> ***/**
[<text position b21>]
<left curly bracket>
[<variable definitions> <end>] <non terminating statements> <end>*
<right curly bracket>
<end>

Additional information

See also <task symbol a49> that describes an alternate syntax to represent task symbols.

Example

```
/* CIF Task (800,550) */
TASK myVariable := 0;"
```

7.5 CIF-GR Syntax – SDL-CIF B rules

The SDL-CIF B rules are utility rules referenced from the A rules and do not by themselves correspond to standalone symbols.

7.5.1 B1 diagram parts

<diagram parts b1> ::=
{ <page declaration b2>+ }

Additional information

There should be one <page declaration b2> for each page in the diagram.

7.5.2 B2 page declaration

<page declaration b2> ::=
/* CIF Page <page name> <size point b22> ***/**
[<frame declaration b11>]
[<diagram heading text position b21>]
[<page text position b19>]
[<package use symbol b14>]

Additional information

If a <frame declaration b11> is not given, the frame has the position (0,0) and the same size as the page. This page becomes the current page and remains the current page until either a new <page declaration b2> or a <page switch a21> is encountered.

The <page text position b19> defines the upper right corner of the surrounding rectangle of the text.

The <package use symbol b14> should only be given if the package use symbol is visible in SDL-GR.

Example

```
    /* CIF Page 1 (1900,2300) */
    /* CIF Frame (100,250),(1700,1950) */
```

7.5.3 B3 gate constraint symbol

<gate constraint symbol b3> ::=
<block symbol rectangle b12> | <process symbol rectangle b13>

7.5.4 B4 first signallist text position

<first signallist text position b4> ::=
/* CIF TextPosition <point b22> **SignalList1 */**

Additional information

This text position should be tied to the first signal list mentioned in the SDL-PR specification.

Example

```
/* CIF TextPosition (800,550) SignalList1 */
```

7.5.5 B5 second signallist text position

<second signallist text position b5> ::=
/* CIF TextPosition <point b22> **SignalList2 */**

Additional information

This text position should be tied to the second signal list mentioned in the SDL-PR specification.

7.5.6 B6 first arrow position

<first arrow position b6> ::=
/* CIF Arrow1Position <point b22> ***/**

Additional information

This arrow position is used for the first channel path mentioned in the SDL-PR for the <channel a22>.

Example

```
/* CIF Arrow1Position (800,550) */
```

7.5.7 B7 second arrow position

<second arrow position b7> ::=
/* CIF Arrow2Position <point b22> ***/**

Additional information

This arrow position is used for the second channel path mentioned in the SDL-PR for the <channel a22>.

7.5.8 B11 frame declaration

<frame declaration b11> ::=
/* CIF Frame <position and size b20> ***/**

7.5.9 B12 block symbol rectangle

<block symbol rectangle b12> ::=
/* CIF Block <position and size b20> ***/**

7.5.10 B13 process symbol rectangle

<process symbol rectangle b13> ::=
/* CIF Process <position and size b20> ***/**

7.5.11 B14 package use symbol

<package use symbol b14> ::=
/* CIF Use <position and size b20> ***/**
[<text position b21>]

Additional information

This SDL-CIF comment is used to specify the position of the package use symbol. The <text position b21> specifies the position of the upper left corner of the bounding box for the package use clauses.

7.5.12 B15 gate reference

<gate reference b15> ::=
/* CIF GateReference [<name>] <connection point b22> ***/**
<text position b21>

Additional information

This SDL-CIF comment is used to specify the text position for a gate reference. The connection point is the point where the gate reference connects to channels. Either the gate name is specified in the rule or, if not specified, the name of the gate is found in the SDL-PR for connected channels. Read more about gates and gate references in <channel a22>. See also the example. Use this SDL-CIF comment for connections via gates. Use <connect a25> for direct connections between channels without gates.

Example

```
/* CIF GateReference (800,550) */
/* CIF TextPosition (750,500) */
```

7.5.13 B16 line

<line b16> ::=
/* CIF Line <pointlist b18> ***/**

7.5.14 B17 dashed line

<dashed line b17> ::=
/* CIF Line <pointlist b18> **Dashed */**

7.5.15 B18 pointlist

<pointlist b18> ::=
<point b22> {, <point b22> }+

7.5.16 B19 page text position

<page text position b19> ::=
/* CIF TextPosition <point b22> **PageName */**

7.5.17 B20 position and size

<position and size b20> ::=
<position point b22> [, <size point b22>]

Additional information

It is allowed to omit the size point if a default size is defined earlier with <default size a20>. The position point is the upper left corner of the surrounding rectangle if nothing else is specified in a higher rule.

7.5.18 B21 text position

<text position b21> ::=
/* CIF TextPosition <point b22> ***/**

Additional information

The point defines the upper left corner of the surrounding rectangle of the text if nothing else is specified in a higher rule. The width and height of the text is not defined.

Example

```
/* CIF TextPosition (800,550) */
```

7.5.19 B22 point

<point b22> ::=
(<integer name>, <integer name> **)**

7.5.20 B23 id text position

<id text position b23> ::=
/* CIF TextPosition <point b22> **TypeRefId */**

Additional information

This text position defines the upper left corner of the surrounding rectangle of the text for the identifier part of a type reference symbol.

7.5.21 B39 in signal list position

<in signal list position b39> ::=
/* CIF TextPosition <point b22> **In */**

Additional information

This text position defines the upper left corner of the surrounding rectangle of the text for the in signal list associated with a gate symbol.

7.5.22 B40 out signal list position

<out signal list position b40> ::=
/* CIF TextPosition <point b22> **Out */**

Additional information

This text position defines the upper left corner of the surrounding rectangle of the text for the out signal list associated with a gate symbol.

7.6 Tool-specific SDL-CIF comments

7.6.1 C0 tool-specific SDL-CIF comment

<tool specific cif comments c0> ::=
/* CIF [**Keep**] **Specific** <tool name> <tool specific information character string> ***/**

Additional information

If **Keep** is given, this tool-specific SDL-CIF comment should be kept if the current SDL-CIF object is edited. If **Keep** is omitted, this SDL-CIF comment should be removed when the current SDL-CIF object is edited.

A tool-specific SDL-CIF comment should be associated with an A rule (equal to SDL-CIF object). The tool-specific SDL-CIF comment should be placed between the SDL-CIF comments and the SDL-PR constructs associated with that A rule. The order between tool-specific SDL-CIF comments associated with the same A rule is undefined.

Informal example (not correct SDL-CIF and not correct SDL-PR)

```
/* CIF 'The CIF comment associated with rule Ax'*/
/* CIF Specific mySDLTool 'This information is understood by mySDLTool.
Tools other than mySDLTool may or may not understand this information.
This tool-specific CIF comment is associated with rule Ax'*/
'This is the SDL-PR associated with A rule number X';
```

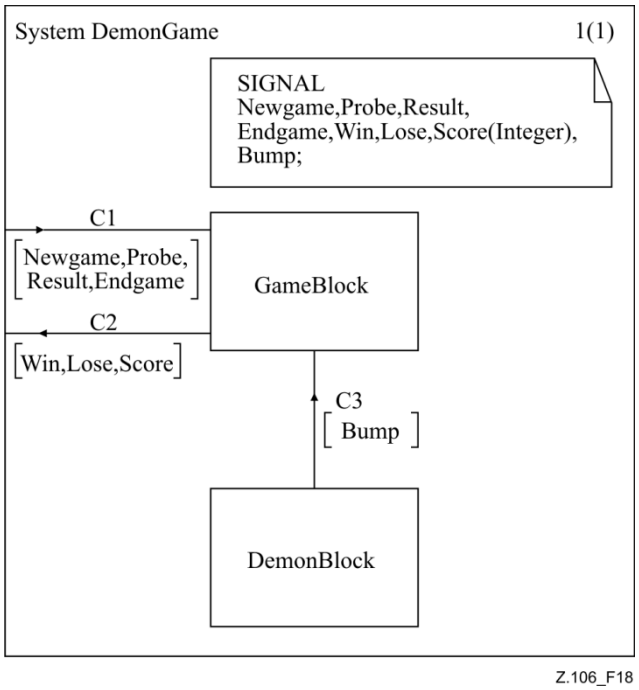
8 Examples

This clause shows some SDL-GR examples and the corresponding SDL-CIF. First, three complete (but small) diagrams are presented, then some tricky SDL-GR constructs are presented and discussed.

Note that most of the examples given here are not nested diagrams, i.e., the SDL-PR uses the keyword REFERENCED. It is of course also allowed in SDL-CIF to express nested diagrams, where the SDL-PR does not have the keyword REFERENCED.

8.1 DemonGame

8.1.1 System DemonGame

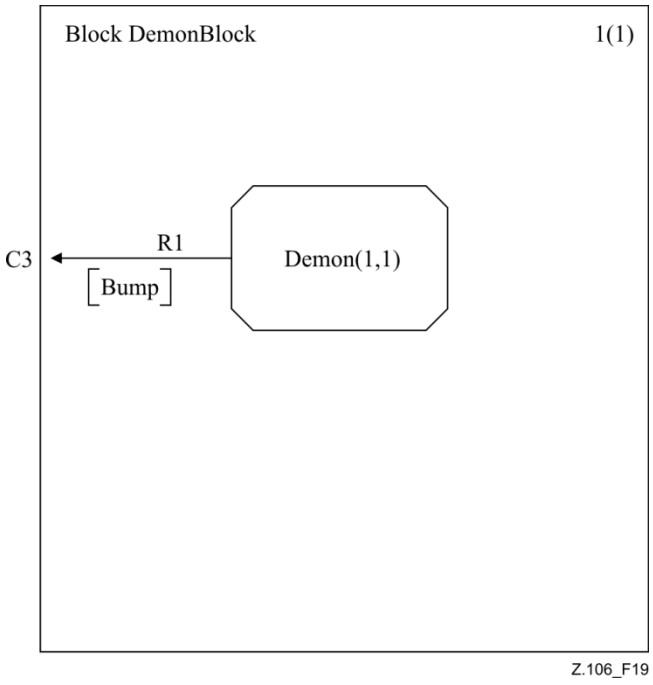



```
     /* CIF SystemDiagram */
     /* CIF Page 1 (1150,1000) */
     System DemonGame;
     /*  CIF  Specific  mySDLTool  Page  1
NoGrid */
     /* CIF DefaultSize (300,200) */
     /* CIF Text (400,100),(600,100) */
     SIGNAL
     Newgame,Probe,Result,
     Endgame,Win,Lose,Score(Integer),Bump;
     /* CIF End Text */
     /* CIF Channel (550,700),(550,500) */
     /* CIF TextPosition (575,551) */
     /*    CIF    TextPosition    (575,600)
SignalList1 */
     channel C3
     from DemonBlock to GameBlock
     with Bump;
     endchannel C3;
     /* CIF Channel (400,475),(0,475) */
     /* CIF TextPosition (150,425) */
     /*    CIF    TextPosition    (62,500)
SignalList1 */
     channel C2
     from GameBlock to env
     with Win,Lose,Score;
     endchannel C2;
     /* CIF Channel (0,325),(400,325) */
     /* CIF TextPosition (212,250) */
     /*    CIF    TextPosition    (50,350)
SignalList1 */
     channel C1
     from env to GameBlock
     with Newgame,Probe,
     Result,Endgame;
     endchannel C1;
     /* CIF BlockSymbol (400,700) */
     block DemonBlock referenced;
     /* CIF BlockSymbol (400,300) */
     block GameBlock referenced;
     /* CIF End SystemDiagram */
     endsystem DemonGame;
```

SDL-GR **SDL-CIF**

Figure 18 – System DemonGame

8.1.2 Block DemonBlock

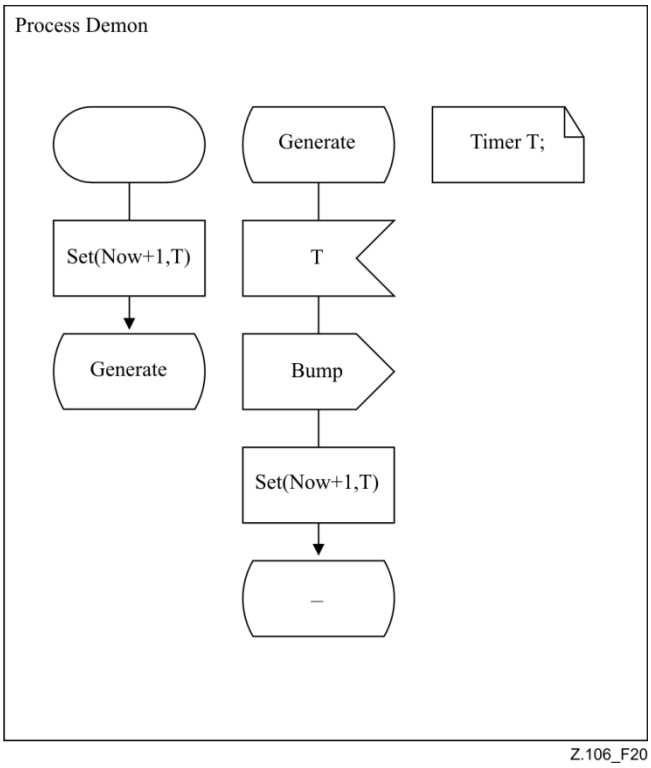



```
     /* CIF BlockDiagram */
     /* CIF Page 1 (1000,1000) */
     /*  CIF  Frame  (100,100),(800,800)
*/
     Block DemonBlock;
     /* CIF Specific mySDLTool Page 1
NoGrid */
     /* CIF DefaultSize (300,200) */
     /* CIF Connect */
     /* CIF TextPosition (25,300) */
     Connect C3 and R1;
     /*         CIF         SignalRoute
(250,350),(0,350) */
     /* CIF TextPosition (150,300) */
     /*   CIF   TextPosition   (75,375)
SignalList1 */
     signalroute R1
     from Demon to env with Bump;
     /* CIF Process (250,250) */
     process Demon (1,1) referenced;
     /* CIF End BlockDiagram */
     endblock DemonBlock;
```

SDL-GR **SDL-CIF**

Figure 19 – Block DemonBlock

8.1.3 Process Demon




```
/* CIF ProcessDiagram */
/* CIF Page 1 (1400,1000) */
Process Demon;
/* CIF DefaultSize (200,100) */
/* CIF Text (800,100) */
Timer T;
/* CIF End Text */
/* CIF Start (300,100) */
start;
/* CIF Set (300,250) */
Set(Now+1,  T);
/* CIF NextState (300,400) */
nextstate Generate;
/* CIF State (550,100) */
state Generate;
/* CIF Input (550,250) */
input T;
/* CIF Output (550,400) */
output Bump;
/* CIF Set (550,550) */
Set(Now+1,  T);
/* CIF NextState (550,700) */
nextstate -;
/* CIF End ProcessDiagram */
endprocess Demon;
```

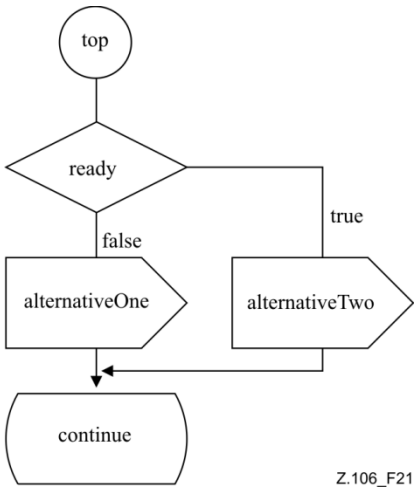
SDL-GR **SDL-CIF**

Figure 20 – Process Demon

8.2 Tricky SDL-2010 constructs

8.2.1 Joining flow lines 1

Note that the pointlist for the flow line after the output symbol alternativeTwo has four points, two of them on the border of symbols.




```
/* CIF Label (50,100),(100,100) */
top:
  /* CIF Line (100,200),(100,300) */
  /* CIF Decision (0,300) */
  decision ready;
  /* CIF Answer */
  /* CIF Line (100,400),(100,500) */
  (false):
    /* CIF Output (0,500) */
    output alternativeOne;
  /* CIF Line (100,600),(100,700) */
  /* CIF Answer */
  /*              CIF              Line
(200,350),(400,350),(400,500) */
  (true):
    /* CIF Output (300,500) */
    output alternativeTwo;
  /*              CIF              Line
(400,600),(400,650),(100,650), (100,700) */
  /* CIF End Decision */
  enddecision;
  /* CIF NextState (0,700) */
  nextstate continue;
```

Figure 21 – Joining flow lines

8.2.2 Joining flow lines 2

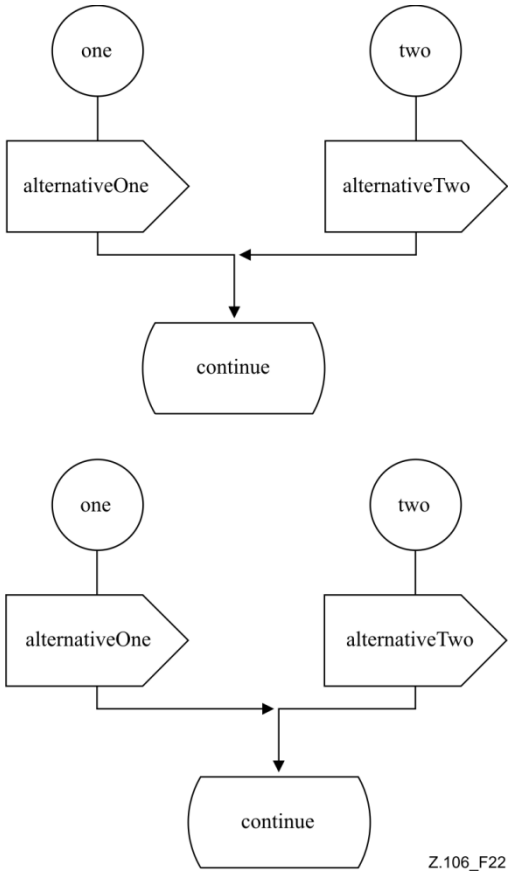
This example is very similar to the previous one. In fact it is the same SDL-GR, but the SDL-PR is expressed differently. Note that the two joining flow lines are placed close to the symbol they are coming from in the SDL-PR below.

```
/* CIF Label (50,100),(100,100) */
top:
/* CIF Line (100,200),(100,300) */
/* CIF Decision (0,300) */
decision ready;
/* CIF Answer */
/* CIF Line (100,400),(100,500) */
(false):
  /* CIF Output (0,500) */
  output alternativeOne;
  /* CIF Line (100,600),(100,700) */
  /* CIF Label Invisible */
  grs0:
  /* CIF NextState (0,700) */
  nextstate continue;
/* CIF Answer */
/* CIF Line (200,350),(400,350),(400,500) */
(true):
  /* CIF Output (300,500) */
  output alternativeTwo;
  /* CIF Line (400,600),(400,650),(100,650),(100,700) */
```

```
  /* CIF Join Invisible */
  join grs0;
/* CIF End Decision */
enddecision;
```

8.2.3 Joining flow lines 3

SDL-CIF does not define which flow line is joining which. The following two SDL-GR examples produce the same SDL-CIF. (Example of SDL-PR without flow lines.)

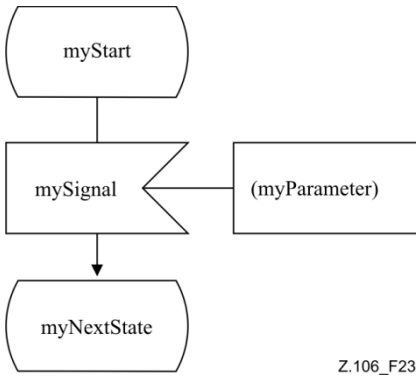



```
/* CIF Label (150,100),(100,100)
*/
connection
  one:
  /* CIF Output (100,300) */
  output alternativeOne;
  /* CIF Label Invisible */
  grs0:
  /* CIF NextState (250,500) */
  nextstate continue;
/* CIF End Label */
endconnection one;
/*  CIF  Label  (450,100),(100,100)
*/
connection
  two:
  /* CIF Output (400,300) */
  output alternativeTwo;
  /* CIF Join Invisible */
  join grs0;
/* CIF End Label */
endconnection two;
```

Figure 22 – Joining flow lines

8.2.4 Lines and enclosing rectangles

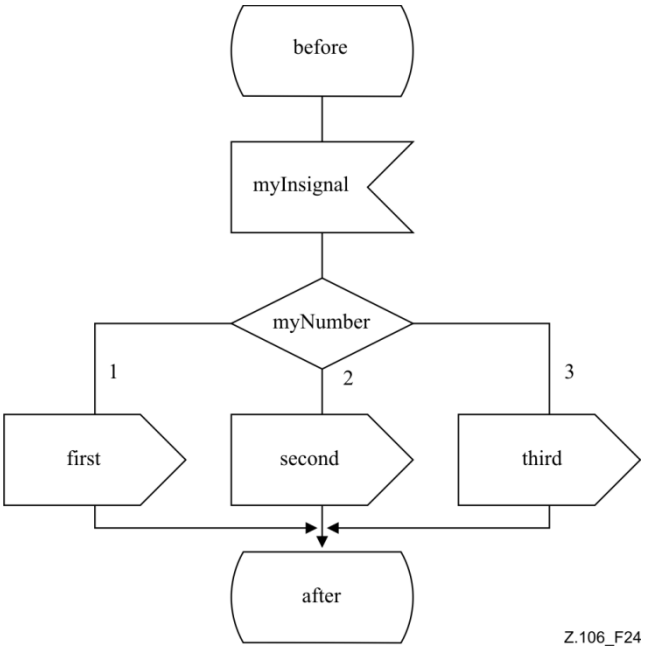
Note that the text extension line is described as if its endpoints were on the border of the surrounding rectangles of the attached symbols, even if this is not completely true in SDL-GR. (The endpoint on the input symbol is inside the surrounding rectangle.) This rule applies to all lines/channels connected to symbols that are not rectangular shaped (stop symbol, process symbol...).




```
/* CIF State (100,100) */
state myStart;
  /* Input (100,300) */
  input mySignal
/* CIF TextExtension (400,300) */
/* CIF Line (400,350),(300,350) */
(myParameter)
/* CIF End TextExtension */
;
    /* CIF NextState (100,500) */
    nextstate myNextState;
```

Figure 23 – Lines and enclosing rectangles

8.2.5 Answer flow lines after decision

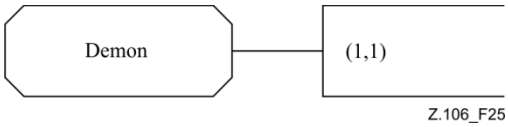



```
/* CIF State (400,100) */
state before;
  /* CIF Input (400,300) */
  input myInsignal;
    /* CIF Decision (400,500) */
    decision myNumber;
    /*     CIF     Answer     Left
InvisibleBrackets */
    (1):
      /* CIF Output (100,700) */
      output First;
    /*         CIF         Answer
InvisibleBrackets */
    (2):
      /* CIF Output (400,700) */
      output second;
    /*     CIF     Answer     Right
InvisibleBrackets */
    (3):
      /* CIF Output (700,700) */
      output third;
    /* CIF End Decision */
    enddecision;
    /* CIF NextState (400,900) */
    nextstate after;
/* CIF End Estate */
endstate;
```

Figure 24 – Answer flow lines

8.2.6 Text extension

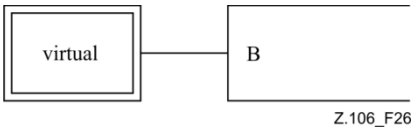
This is allowed:




```
/* CIF Process (800,550) */
PROCESS Demon
/* CIF TextExtension (1100,550) */
(1,1)
/* CIF TextExtensionEnd */
REFERENCED;
```

Figure 25 – Text extension

This is also allowed:

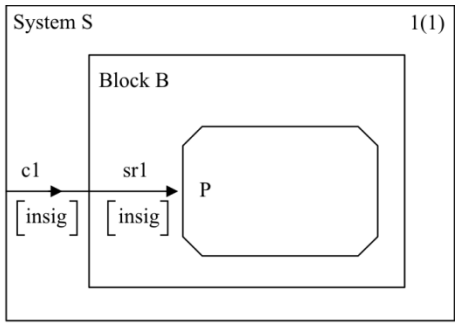



```
/* CIF BlockType (800,550) */
virtual BLOCK TYPE
/* CIF TextExtension (1100,550) */
B
/* CIF TextExtensionEnd */
REFERENCED;
```

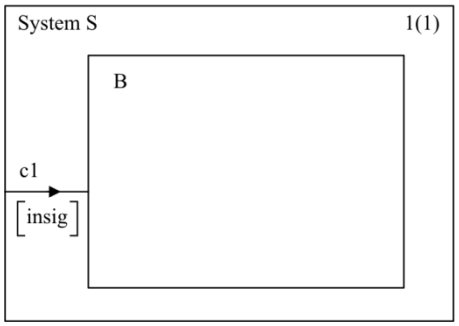
Figure 26 – Text extension

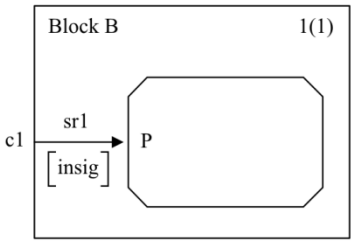
8.2.7 Nested diagrams

Nested diagrams are not allowed in SDL-2010.




```
/* CIF SystemDiagram */
/* CIF Page 1 (600,500) */
System S;
/* CIF Channel (0,250),(100,250) */
CHANNEL c1
FROM ENV TO B WITH insig;
/* CIF Block (100,100),(400,300) */
BLOCK B REFERENCED;
/* CIF End SystemDiagram */
ENDSYSTEM S;
```



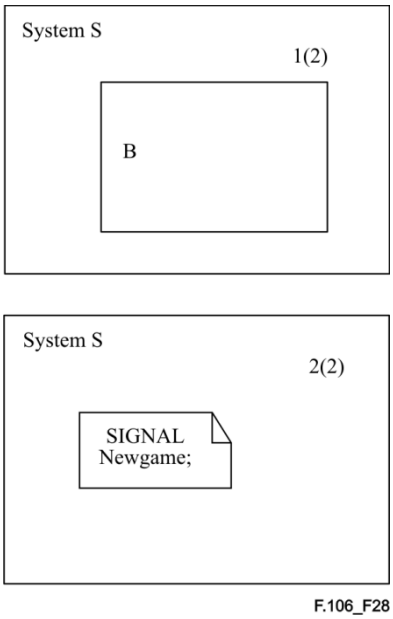



```
/* CIF BlockDiagram */
/* CIF Page 1 (600,500) */
/* CIF Frame (100,100),(400,300) */
Block B;
/* CIF Channel (100,250),(200,250) */
CHANNEL sr1
FROM ENV TO P WITH insig;
/* CIF Connect */
CONNECT c1 AND sr1;
/* CIF Process (200,200),(200,100) */
PROCESS P REFERENCED;
/* CIF End BlockDiagram */
ENDBLOCK B;
```

Z.106_F27

Figure 27 – Logically nested diagram for block B

8.2.8 Many pages




```
/* CIF SystemDiagram */
/* CIF Page 2 (600,300) */
/* CIF Page 1 (600,300) */
System S;
/* CIF DefaultSize (200,100) */
/* CIF Block (200,100) */
BLOCK B REFERENCED;
/* CIF CurrentPage 2 */
/* CIF Text (200,100) */
SIGNAL
NewGame;
/* CIF End Text */
/* CIF End SystemDiagram */
ENDSYSTEM S;
```

Figure 28

8.3 Situations SDL-CIF is not able to handle

Figure 29

SDL-CIF is not able to handle this properly. The SDL-CIF code would be:

```
/* CIF Process (800,550) Dashed myProcess */
```

The comment in the SDL-GR is lost. The consequence is that SDL-CIF is not usable as a storage format for all legal SDL-GR diagrams. To manage this example properly, the complete text within the symbol has to be stored inside the SDL-CIF comment. This would mean that escape characters would have to be introduced to manage start of comment, end of comment, start of text and end of text tokens/characters within the text. However, this has been left out of SDL-CIF to avoid too much complexity.

This is one example from the class of problems that arise because SDL-PR is not matching SDL-GR exactly. Problems with expressing textual comments in SDL-PR can also be found in connection with, for instance, connect, gates and macros.

Here is another example of a problem with comments. This time it is a comment symbol. How would this be expressed in SDL-PR?

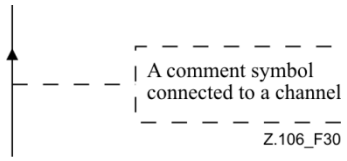


Figure 30

Here is one more SDL-GR example (possibly not legal SDL-GR) that neither SDL-CIF nor SDL-PR at the moment can handle well. (The most reasonable thing to do when converting this to SDL-PR is to duplicate the text extension/comment symbol. What should then happen when this is converted back to SDL-GR?)

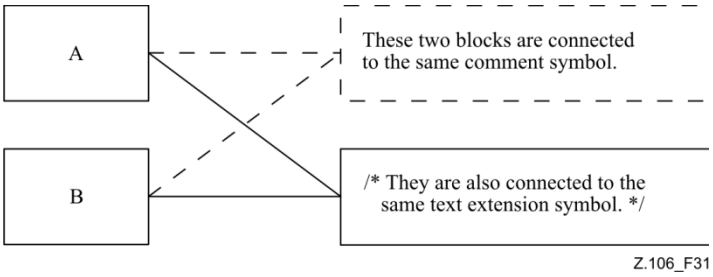


Figure 31

There is one situation that SDL-CIF cannot handle because of the construction of SDL-CIF itself. It is if the SDL-GR contains something that resembles SDL-CIF:

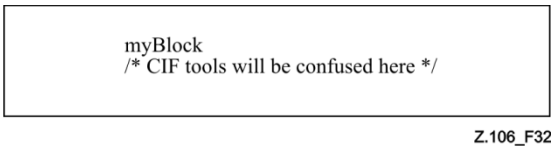


Figure 32

9 SDL-CIF conformance criteria

9.1 About tools reading a SDL-CIF file

A tool reading an SDL-CIF file should try to be forgiving instead of following the rules strictly.

Example 1: The SDL-CIF file states that the current page is X, but there is no page X in the current diagram. The tool should issue a warning and ignore the erroneous current page SDL-CIF comment.

Example 2: The SDL-CIF file describes a flow line that does not end on the border of a symbol in both ends. The tool should issue a warning and ignore the erroneous flow line SDL-CIF comment.

9.2 Automatic versus forced layout

When an SDL-CIF file is exported from a tool, it contains positioning information that relates to the layout of the SDL-2010 diagrams. When another tool imports this information, it will make use of the layout information in the SDL-CIF file. If the tool does not use the layout information and uses an automatic layout mechanism, then it conforms to the level 1 SDL-CIF for import only. Tools that support SDL-CIF should clearly indicate whether they use forced layout when importing SDL-CIF and how they support the preservation of graphical information and to which SDL-CIF levels they conform for export and for import.

9.3 Retention and use of tool-specific information

The SDL-CIF has been intentionally defined to provide tool-specific information that another tool is able to either use or ignore. ITU-T recommends that, where tool vendors use tool-specific information, they provide this information publicly to avoid a clash with other tools. Examples have been provided in this Recommendation and it is possible future SDL-CIF versions will include additional tool-specific information to enable other tool vendors to support the import and possible preservation of this information. No licence fees shall be applicable to any tool-specific information that uses the facilities for tool-specific information provided by this Recommendation.

Appendix I

Tool-specific SDL-CIF comments

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

I.1 Maintenance of SDL-CIF

As it is impossible to foresee in SDL-CIF all potential requests coming from makers of tools regarding tool-specific directives, the syntax has been made open to continuously integrate new tool-specific directives.

Initially, names of new tools are approved by experts working within ITU-T to ensure that there is no name conflict.

In the perspective of enriching SDL-CIF, makers of tools who have created new tool-specific directives should propose them to the responsible experts working within ITU-T. In this way, these directives have a chance to be introduced in this Recommendation, as new non-specific recommended directives, either mandatory or optional.

Successful and unsuccessful implementations of SDL-CIF should be reported to the responsible experts working within ITU-T, in order to adjust to mandatory/optional classification of directives.

An up-to-date list of tool-specific SDL-CIF comments in use by various tools is available on the ITU website. In addition, this appendix provides an initial list of tool-specific SDL-CIF comments that were known when this appendix was produced.

I.2 Current tool-specific SDL-CIF comments

The tool-specific SDL-CIF comments defined below are not a part of the SDL-CIF standard. They constitute the currently known tool-specific SDL-CIF comments.

How a tool should handle tool-specific SDL-CIF comments is defined in clause 7.6.

In this appendix an imaginary SDL-2010 tool will be used, which has <tool name> equal to mySDLTool; a tool developer should substitute mySDLTool for an appropriate name that uniquely identifies the tool in question.

The tool-specific rules listed below are intentionally defined but not used:
<tool specific cif comments c0>
<tool version number c1>
<original file c2>
<page details specification c3>
<fixed size c4>

I.2.1 Placement of tool-specific SDL-CIF comments

This topic is discussed in the additional information associated with <tool specific cif comments c0> in clause 7.6.1.

I.2.2 Example

In the example below, many tool-specific SDL-CIF comments are associated with the SDL-CIF rule <system diagram start a6>. Detailed information for each tool-specific SDL-CIF comment mentioned in this example can be found later in this appendix.

```
/* CIF SystemDiagram */
/* CIF Page 1 (1900,2300) */
/* CIF Page 2 (1900,2300) */
/* CIF Frame (100,100),(1700,2100) */
/* CIF Specific mySDLTool Version 1.0 */
```

```
/* CIF Specific mySDLTool Page 1 Scale 200 AutoNumbered
FixedHeadingSize (200,100) */
/* CIF Specific mySDLTool Page 2 Scale 200 AutoNumbered */
/* CIF Specific mySDLTool OriginalFileName 'mysystem.ssy'*/
SYSTEM mySystem;
/* CIF CurrentPage 1 */
/* CIF Text (800,550),(200,100) */
/* CIF Specific mySDLTool FixedSize (200,100) */
dcl
  MyNo Integer;
timer
  DoorTimer;
/* CIF End Text */
...
```

I.2.3 C1 tool version number

<tool version number c1> ::=
/* CIF Specific mySDLTool Version x.y */
<diagram description a2>

Association

This tool-specific SDL-CIF comment is associated with an A rule mentioned in the rule for <diagram start a3>.

Example (where this tool-specific SDL-CIF rule is associated with <system diagram start a6>)

```
/* CIF SystemDiagram */
/* CIF Page 1 (1900,2300) */
/* CIF Frame (100,100),(1700,2100) */
/* CIF Specific mySDLTool Version 1.2 */
SYSTEM mySystem;
```

I.2.4 C2 original file

<original file c2> ::=
/* CIF Specific mySDLTool OriginalFileName <file name character string> literal ***/**
<diagram start a3>

Association

This tool-specific SDL-CIF comment is associated with an A rule mentioned in the rule for <diagram start a3>.

Additional information

This comment is used to remember the file name of the binary file that was converted to SDL-CIF. When the SDL-CIF file is converted to a binary file again, the same name can be reused. Only the file name (a.ssy) will be given in <file name character string> literal, not the complete path (/home/lat/a.ssy).

Example (where this tool-specific SDL-CIF rule is associated with <system diagram start a6>)

```
/* CIF SystemDiagram */
/* CIF Page 1 (1900,2300) */
/* CIF Frame (100,100),(1700,2100) */
/* CIF Specific mySDLTool OriginalFileName 'mysystem.ssy' */
SYSTEM mySystem;
```

I.2.5 C3 page details specification

<page details specification c3> ::=
/* CIF Specific mySDLTool Page <page number> [**ShowMeFirst**] [**Scale** <integer name>] [**Grid** <point b22>]
[**AutoNumbered**] [**FixedHeadingSize** [<fixed size point b22>]] ***/**

Association

This tool-specific SDL-CIF comment is associated with an A rule mentioned in the rule for <diagram start a3>. This tool-specific SDL-CIF comment is optional for each <page declaration b2> in the diagram.

Additional information

This tool-specific SDL-CIF comment is used to give additional information for an already defined page:

– An indication of whether this page should pop up as the default page when the diagram is loaded into an editor (The keyword ShowMeFirst is given.) or not (The keyword ShowMeFirst is not given.).

– Specification of the scale used in the SDL-2010 editor when presenting the diagram. Scale is expressed as per cent of normal size. If scale is not given, 100% is used.

– Specification of the grid size for the page. If grid is not given, grid will be (50,50) which is the same as no grid.

– An indication of whether autonumbering should be used (The keyword AutoNumbered is given.) or not (The keyword AutoNumbered is not given.). AutoNumbered means that the tool will keep track of page numbering in the following way: initially, the first declared autonumbered page will get the number "1", the second declared autonumbered page will get the number "2"... Later, when the user adds an autonumbered page before page "2", the new page will get the number "2" and all old autonumbered pages after page "1" will get their name increased by one. A unique <page name character string> literal shall be given in <page declaration b2> for the page (even if the page is autonumbered) to be able to refer to the page from other SDL-CIF comments.

In addition, if FixedHeadingSize is given, this tool-specific SDL-CIF comment states that the additional heading symbol should be given a fixed size and not be adjusted to the size of the text. This means that the complete text will not be visible. The user has to double click on the additional heading symbol to expand it to the larger size. The SDL-2010 tool uses this comment for additional heading symbols that the user has "collapsed" with a double click.

Example (where this tool-specific SDL-CIF rule is associated with <system diagram start a6>)

```
/* CIF SystemDiagram */
/* CIF Page 1 (1900,2300) */
/* CIF Page 2 (1900,2300) */
/* CIF Frame (100,100),(1700,2100) */
/* CIF Specific mySDLTool Page 1 Scale 200 AutoNumbered */
/* CIF Specific mySDLTool Page 2 Scale 200 AutoNumbered */
SYSTEM mySystem;
/* CIF Text (800,550) */
SIGNAL Newgame;
/* CIF End Text */
```

I.2.6 C4 fixed size

<fixed size c4> ::=
/* CIF Specific mySDLTool FixedSize [<fixed size point b22>] ***/**

Association

This tool-specific SDL-CIF comment is associated with <text symbol a68>.

Additional information

This tool-specific SDL-CIF comment states that although the symbol just defined in an SDL-CIF comment was given a size (a size that makes the complete text visible), a fixed smaller size is used

to render the symbol. This means that the complete text will not be visible. The user has to double click on the symbol to expand it to the larger size. The SDL-2010 tool uses this comment for text symbols that the user has "collapsed" with a double click.

Example

```
/* CIF Text (800,550),(200,200) */
/* CIF Specific mySDLTool FixedSize (200,100) */
dcl
  MyNo Integer;
timer
  DoorTimer;
/* CIF End Text */
```

SERIES OF ITU-T RECOMMENDATIONS

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

Printed in Switzerland
Geneva, 2019