INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.106

(08/2002)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

Common interchange format for SDL

ITU-T Recommendation Z.106

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
Extended Object Definition Language (eODL)	Z.130–Z.139
Tree and Tabular Combined Notation (TTCN)	Z.140–Z.149
User Requirements Notation (URN)	Z.150–Z.159
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-computer interfaces for the management of telecommunications networks	Z.360–Z.369
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	
Distributed processing environment	Z.600–Z.609

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

ITU-T Recommendation Z.106

Common interchange format for SDL

Summary

This Recommendation defines the Common Interchange Format (CIF) of Specification and Description Language (ITU-T Rec. Z.100 – SDL). The CIF is intended for the interchange of graphical SDL specifications (SDL-GR) made on different tools that do not use the same storage format. Prior to the definition of CIF, the textual representation of SDL (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 CIF, this disadvantage is reduced to a minimum, as it contains most of the graphical information. The 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 tool vendors are encouraged to provide facilities for importing and exporting SDL-CIF.

This Recommendation defines how SDL descriptions can be stored in order to be interchanged between tools coming from different vendors. It does not take into account the MSC notation. SDL-CIF is an extension to SDL. CIF is based on the SDL-PR syntax, the textual Phrase Representation of SDL also defined in this Recommendation. CIF can be read and written by tools as well as users. All the constructs available in SDL can be expressed in graphical form or in the purely textual SDL-PR form. Constraints on graphical presentation are expressed in 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 a CIF-compliant tool to extend the format by adding specific information. SDL-CIF is also easily implementable 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 SDL. Before 2002, SDL-PR was published as part of 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 CIF and allows the interchange of syntactically complete SDL descriptions, usually as a single file per system. Conformance to SDL-PR requires the model to conform to the corresponding semantics defined in ITU-T Rec. Z.100.

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

Source

ITU-T Recommendation Z.106 was revised by ITU-T Study Group 17 (2001-2004) and approved under the WTSA Resolution 1 procedure on 6 August 2002.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. 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.

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, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

Page

Background

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

Until the time this Recommendation had been proposed, the only way to interchange specifications in SDL had been to interchange SDL-PR (the textual representation of SDL). This has led to the loss of graphical information. Though not necessary from the formal point of view, 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 SDL specifications without the loss of graphical information.

ITU-T Recommendation Z.106

Common interchange format for SDL

1 Scope

This Recommendation defines the Common Interchange Format for specifications written in Specification and Description Language (SDL) [1]. It is intended for tool vendors as an export and import format to allow the interchange of SDL specifications with tools offered by other tool vendors. Even though the format allows writing specifications in CIF directly, it is not intended for this purpose but rather should be generated by an existing SDL specification written according to [1].

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.

[1] ITU-T Recommendation Z.100 (2002), *Specification and Description Language (SDL)*.

3 Abbreviations

This Recommendation uses the following abbreviations:

CIF Common Interchange Format

SDL CCITT Specification and Description Language

4 Notation used in this Recommendation

For the definition of properties and syntaxes of CIF, the metalanguage defined in [1], § 5.4.2, has been used with the following addition. When the first word of a non-terminal symbol is separated from the following words by a colon, this symbol is a reference to another CIF rule. Non-terminal symbols refer to non-terminals of ITU-T Rec. Z.100 or to non-terminals introduced in this Recommendation.

5 Level 0 CIF (SDL-PR)

5.1 General principles

CIF level 0 introduces an additional syntactic form that can be used when representing a system: the textual Phrase Representation (SDL-PR). SDL-PR can be used instead of the graphic representation of [1], which shall be referred to as SDL-GR in this document. As both are concrete representations of the same SDL, 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 can 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 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.

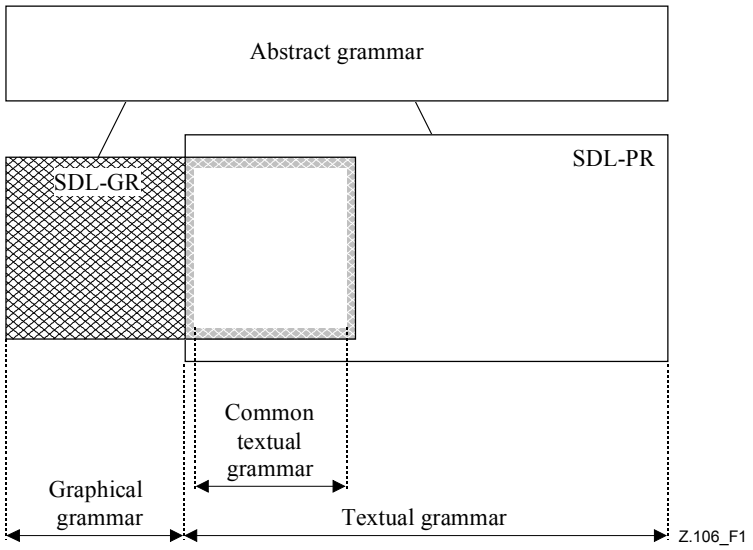


Figure 1/Z.106 – SDL grammars

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. 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 specified in the extended Backus-Naur Form of syntax description defined in [1], 5.4.2.

The textual syntax may be followed by paragraphs defining the static conditions which must be satisfied in a well-formed text and which can be checked without interpretation of an instance.

In many cases, there is a simple relationship between the concrete and abstract syntax, because the concrete syntax rule is simply represented by a single rule in the abstract syntax. When the same name is used in the abstract and concrete syntax in order to signify that they represent the same concept, then the text "<x> in the concrete syntax represents *X* in the abstract syntax" is implied in the language description and is often omitted. In this context, case is ignored but underlined semantic sub-categories (see [1], 5.4.2) 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 (see [1], 5.3.2).

NOTE – In the following, the titles of the clauses correspond to the titles of the matching clauses of [1].

5.2 General rules

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

5.3 Organization of SDL specifications

5.3.1 Framework

The starting production rule of ITU-T Rec. Z.100, <sdl specification> (see [1], 7.1), is replaced by the following production:

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

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>
| <signal reference>
| <signal list definition>
| <remote variable definition>
| <textual data definition>
| <data definition>
| <data type reference>
| <textual procedure definition>
| <procedure definition>

| <procedure reference>
| <remote procedure definition>
| <composite state type definition>
| <composite state type reference>
| <exception definition>
| <select definition>
| <macro definition>
| <interface reference>
| <association>

5.3.3 Referenced definition

The production rule of ITU-T Rec. Z.100, <referenced definition> (see [1], 7.3), is replaced by the following production:

<referenced definition>: :=
<definition>

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

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>

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>

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>

5.4.1.2 Composite state type

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

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> must 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 definition>: :=
state aggregation <typebased state partition heading> <end>

5.4.1.3 Type references

The production rule of ITU-T Rec. Z.100, <type referenced properties> (see [1], 8.1.5), is replaced by the following production:

<type reference properties>: :=
[**with** { <attribute property> <end> }+]
[**with** { <behaviour property> <end> }+]
referenced <end>

5.4.1.4 Gate

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

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

<textual interface gate definition>: :=
gate { **in** | **out** } **with** <interface identifier> <end>

adding may only be specified 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.

5.4.2 Associations

<association>: :=
association [<association name>] <association kind> <end>

<association kind>::=
<association not bound kind>
| <association end bound kind>
| <association two ends bound kind>
| <composition not bound kind>
| <composition part end bound kind>
| <composition composite end bound kind>
| <composition two ends bound kind>
| <aggregation not bound kind>
| <aggregation part end bound kind>
| <aggregation aggregate end bound kind>
| <aggregation two ends bound kind>

<association not bound kind>: :=
from <association end> **from** <association end>

<association end bound kind>::=
from <association end> **to** <association end>

<association two ends bound kind>::=
to <association end> **to** <association end>

<composition not bound kind>::=
from <association end> **composition** <association end>

<composition part end bound kind>::=
to <association end> **composition** <association end>

<composition composite end bound kind>: :=
from <association end> **to composition** <association end>

<composition two ends bound kind>::=
to <association end> **to composition** <association end>

<aggregation not bound kind>::=
from <association end> **aggregation** <association end>

<aggregation part end bound kind>::=
to <association end> **aggregation** <association end>

<aggregation aggregate end bound kind>::=
from <association end> **to aggregation** <association end>

<aggregation two ends bound kind>::=
to <association end> **to aggregation** <association end>

<association end>: :=
[<visibility>] [**as** <role name>] [**size** <multiplicity>] [**ordered**]
{ <agent type reference> | <interface reference> | <data type reference> }

If an <association end> identifies an agent type or an interface, the **protected** visibility cannot be used in the other <association end> of the <association>.

If two different <association>s identify the same type, in the <association end>s opposite to this common type the <role name>s (if given) must be different.

There must not be a set of <association>s containing composition such that a type is linked by composition back to itself, either directly or indirectly.

If an <association end> is preceded by the keyword **composition** and identifies a data type or interface, then the other <association end> of the <association> must identify a data type or interface, respectively.

5.5 Agents

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

<agent structure>: :=
[<valid input signal set>]
{ { <entity in agent>
| <channel to channel connection>
| <channel definition>
| <gate in definition>
| <agent definition>
| <agent reference>
| <textual typebased agent definition> }*
[<state partition>]
| { <entity in agent>
| <gate in definition> }*
<agent body> }

The <state partition> must have the same name as the containing agent. It defines the state machine of the agent. If <agent structure> can be understood both as <state partition> and <agent body>, it is interpreted as <agent body>.

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

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

| <agent type definition>
| <agent type reference>
| <association>

<agent body>: :=
[[<on exception>] <start>]
{ <state> | <exception handler> | <free action> } *

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>

5.5.3 Process

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

5.5.4 Procedure

<textual procedure definition>: :=
{<package use clause>}*
<procedure heading> <end>
<entity in procedure>*
[<procedure body>]
endprocedure [[<qualifier>] <procedure name>] <end>

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

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

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

5.6 Communication

5.6.1 Channel

<channel definition>: :=
channel [<channel name>] [**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>]

<via gate>: :=
via <gate>

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

The <channel endpoint> **this** denotes the state machine of the agent directly enclosing the channel definition.

<gate> must be specified if:

a) <channel endpoint> denotes a connection to a <textual typebased agent definition> or a <textual typebased state partition definition> in which case the <gate> must be defined directly in the agent type or state type for that agent or state respectively; or

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

If <gate> is specified, the channel is connected to that gate. The gate and the channel must 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 following conditions must be satisfied:

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

5.6.2 Connection

<channel to channel connection>: :=
connect <external channel identifiers>
and <channel identifiers> <end>

<channel identifiers>: :=
<channel identifier> {, <channel identifier>}*

No channel may be mentioned 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 its <channel identifiers> formed by listing all the internal <channel identifier>s.

NOTE – Because of the **connect** construct, an (external) channel, which can be anonymous in the graphical version of a specification, may need to have a name in the corresponding textual version. This is completely analogous to the case of <merge area>s in process 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> [<on exception>] <transition>

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

5.7.2 State

<state>: :=

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

The optional <state name> ending a <state> may be specified only if the <state list> in the <state> consists of a single <state name> in which case it must be that <state name>.

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

5.7.3 Input

<input part>: :=

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

5.7.4 Priority input

<priority input>: :=
priority input [<virtuality>]
<priority input list> <end> [<on exception>]<transition>

5.7.5 Continuous signal

<continuous signal>: :=
provided [<virtuality>]
<continuous expression> <end>
[**priority** <priority name> <end>] [<on exception>] <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>
[<on exception>] [<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** [<connector name>] <end>]

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

5.7.10 State machine and Composite state

<composite state>: :=
<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 structure>: :=
[<valid input signal set>]
{<gate in definition>}*
<state connection points>*
<entity in composite state>*
{ <composite state body> | <state aggregation body> }

A <composite state structure> shall contain a <state aggregation body> only if it is directly contained in a <state aggregation> or a <composite state type definition> with a <state aggregation type heading>, otherwise it contains a <composite state body>. A <composite state structure> that contains a <state aggregation body> shall not have a <valid input signal set>.

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

<composite state body>: :=
[<on exception>] <start>* { <state> | <exception handler> | <free action> }*

5.7.10.2 State aggregation

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

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

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

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

<state partition connection>: :=
connect <outer entry point> **and** <inner entry point> <end>

<outer entry point>: :=
<composite state identifier> **via** <point>

<inner entry point>: :=
<composite state identifier> **via** <point>

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>
[<on exception>] <exit transition>

<exit transition>: :=
<transition>

5.7.11 Transition

5.7.11.1 Transition body

<transition>: :=
{<transition string> [<terminator>] }
| <terminator>

<transition string>: :=
{<action>}+

<action>: :=
[<label>]
{ <task>
| <output>
| <create request>
| <decision>
| <set>
| <reset>
| <export>
| <procedure call>
| <remote procedure call>
| <transition option> } <end>

<terminator>: :=
[<label>]
{ <return>
| <raise>
| <nextstate>
| <join>
| <stop> } <end>

If the <terminator> of a <transition> is omitted, then the last action in the <transition> must contain a terminating <decision> (see 5.7.12.2) or terminating <transition option>, except when a <transition> is contained in a <decision> or <transition option>.

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 must be exactly one <connector name> corresponding to a <join> within the same body. The rule for <agent body> in a type definition is stated in [1], 8.3.1.

5.7.11.2.3 Stop

<stop>: :=
stop

5.7.11.2.4 Return

<return>: :=
return [<return body>] [<end> <on exception>]

5.7.11.2.5 Raise

<raise>: :=
raise <raise body>

5.7.12 Action

5.7.12.1 Task

<task>: :=
task [<comment body>] <left curly bracket> <task body> <right curly bracket>
[<end> <on exception>]

5.7.12.2 Create

<create request>: :=
create <create body> [<end> <on exception>]

5.7.12.3 Procedure call

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

5.7.12.4 Output

<output>: :=
output <output body> [<end> <on exception>]

5.7.12.5 Decision

<decision>: :=
decision <question> <end> [<on exception>]
<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> must be omitted if and only if the <question> consists of the keyword **any**. In this case, no <textual else part> may be present.

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> [<end> <on exception>]

<set>: :=
set <set body> [<end> <on exception>]

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

5.7.14 Exception

5.7.14.1 Exception handler

<exception handler>: :=
exceptionhandler <exception handler list> <end>
[<on exception>]
<handle>*
[**endexceptionhandler** [<exception handler name>] <end>]

5.7.14.2 On-Exception

<on exception>: :=
onexception <exception handler name>

5.7.14.3 Handle

<handle>: :=
handle [<virtuality>] <exception stimulus list> <end>
[<on exception>] <transition>

5.8 Data

5.8.1 Data definitions

<textual data definition>: :=
<textual data type definition>
| <textual interface definition>
| <textual syntype definition>

5.8.1.1 Data type definition

<textual data type definition>: :=
{<package use clause>}*
<type preamble> <data type heading> [<data type specialization>]
<end> [<data type definition body> <data type closing> <end>]

<data type closing>: :=
{ **endvalue** | **endobject** } **type** [<data type name>]

The keyword **value** or **object** and <data type name> in a <data type closing> must be matched by the keywords **endvalue** or **endobject**, respectively, and name in the corresponding <data type heading>, if present.

5.8.1.2 Interface definition

<textual interface definition>: :=
{<package use clause>}*
[<virtuality>] <interface heading>
[<interface specialization>]
<end> <entity in interface>* [<interface use list>]
<interface closing>
| {<package use clause>}*
[<virtuality>] <interface heading>
[<interface specialization>] <end>

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

5.8.1.3 Behaviour of operations

<textual operation definition>: :=
{<package use clause>}*
<operation heading> <end>
<entity in textual operation>*
<operation body>
{ **endoperator** | **endmethod** } [[<qualifier>] <operation name>] <end>

<entity in textual operation>: :=
<textual data definition>
| <data definition>
| <variable definition>
| <exception definition>
| <select definition>
| <macro definition>

<operation body>: :=
[<on exception>] <start> { <free action> | <exception handler> }*

5.8.1.4 Syntypes

<textual syntype definition>: :=
{<package use clause>}*
syntype <syntype name> <equals sign> <parent sort identifier>
{ { <default initialization> [[<end>] [<constraint>] | <constraint> } <end>]
<syntype closing>
| [<syntype closing>] } <end>
| {<package use clause>}*
<type preamble> <data type heading> [<data type specialization>] <end>

<data type definition body> <constraint> <end> <data type closing> <end>

<syntype closing>: :=

endsyntype [<syntype name>]

5.9 Generic system definition

5.9.1 Optional definition

<select definition>: :=

select if (<Boolean simple expression> **)** <end>
{ <agent type definition>
| <agent type reference>
| <agent definition>
| <agent reference>
| <channel definition>
| <signal definition>
| <signal list definition>
| <signal reference>
| <remote variable definition>
| <remote procedure definition>
| <textual data definition>
| <data definition>
| <data type reference>
| <interface reference>
| <composite state type definition>
| <composite state type reference>
| <state partition>
| <timer definition>
| <variable definition>
| <textual procedure definition>
| <procedure definition>
| <procedure reference>
| <channel to channel connection>
| <select definition>
| <macro definition>
| <exception definition> }+

endselect <end>

5.9.2 Optional transition string

<transition option>: :=

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

6 Level 1 CIF (CIF-PR)

6.1 General principles

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 descriptions, while there is a need to interchange descriptions

which are partial or not yet achieved. However, these descriptions must be syntactically correct to be interchanged.

SDL-CIF reuses large parts of the SDL-PR syntax. The production rules of SDL-PR which are reused in SDL-CIF are later just referenced by their name, they are not described again in this Recommendation.

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

6.2 Transferable units of SDL specifications

6.3 CIF-PR syntax

6.3.1 CIF file

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

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

6.3.2 Macro call

SDL macro calls may "appear at any place where a <lexical unit> is allowed" ([1], 6.2.3).

SDL/CIF macro calls can only appear at the place of a task.

The production rule <task> (see 5.7.12.1) is replaced by:

<task>: :=
task <textual task body> [<end> <on exception>]
| <macro call>

7 Level 2 CIF (CIF-GR)

7.1 General principles

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

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 CIF converter has to extract should have an associated CIF comment placed before the SDL-PR construct. This makes it possible for a CIF reading tool to scan for the next CIF comment, extract necessary information from the following SDL-PR and then start looking for the next CIF comment.

Several CIF directives may be associated with the same object.

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

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 SDL editors favour manual (user) layout of symbols while others favour automatic layout. Handling both manual and automatic layout is a complex problem which is difficult to solve when developing an SDL tool.

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

Mandatory directives describe graphical characteristics which cannot 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 SDL editing tools.

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

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

7.2 General principles, 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

Diagrams may be split into pages, as described in [1], 6.6.

However, SDL-CIF files are not structured according to pages, but according to the SDL-PR syntax. Pages are described by CIF comments inserted between some syntax units. A page may consist of information from several non-adjacent syntax units.

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

7.2.3 Classification of information

CIF graphical information can be classified into 4 classes:

– information about symbols which look like graphical lines, usually just called "lines", i.e. channels, flow lines in transitions, and association lines;

– 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 which exist in both a left and right version, a "Left" or "Right" keyword may be 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 CIF. This means that a tool does not have to specify a text position when writing a CIF file. It does also mean that a tool reading a CIF file with a specified text position does not have to use that text position. Here are some guidelines:

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

If a text position is not given in the 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 CIF. A tool maker 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.

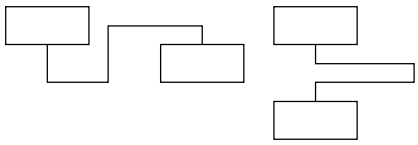
The start and end points of a line are usually connected to other symbols.

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/Z.106

7.2.9 Graphical information not covered by CIF

CIF directives are mainly related to graphical positions and sizes, because it is universal information that can be interchanged 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, font size, colour and line thickness.

They can be given by tool-specific directives.

7.2.10 About nested diagrams

Nested SDL (diagrams within diagrams) is supported by CIF. Tools that do not support nested SDL can convert a diagram within a diagram to a reference symbol and a separate diagram:

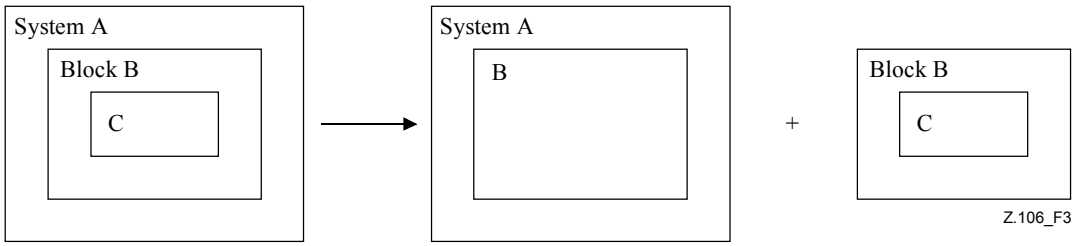


Figure 3/Z.106

7.2.11 About kernel and additional heading

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 CIF. Note that the SDL-GR in the example below is not correct, because according to ITU-T Rec. Z.100 there should not be a symbol around the additional heading text.

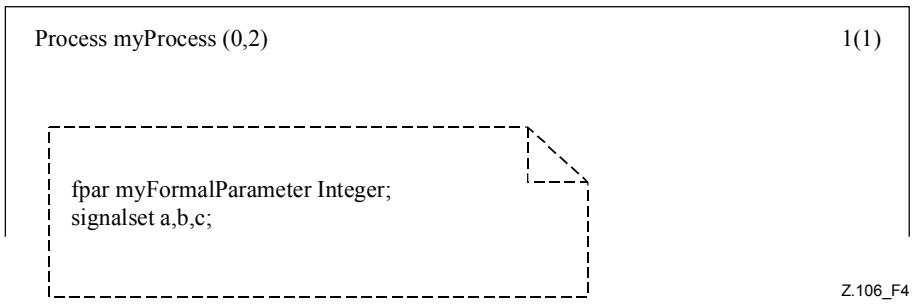


Figure 4/Z.106

A heading text split into kernel heading and additional heading.

7.3 CIF-GR lexical rules

7.3.1 CIF directives

CIF directives are special forms of the 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 a CIF directive, must not contain any other token.

A CIF-GR <note> must 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 CIF file, and are then ignored.

However, when two adjacent SDL 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 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 must 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 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 must 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: CIF A rules

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

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

- A section describing the grammar for the CIF comment and for related SDL-PR information. This section also shows how 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 CIF comments into SDL-PR. A CIF level 2 file is correct if it both satisfies the CIF level 2 grammar (for the definition of CIF comments) and the CIF level 1 grammar.

7.4.1 A1 CIF Description:

{ <diagram description:A2> | }*

Additional information:

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

7.4.2 A2 Diagram Description:

<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:

<specification area start: A5B> | <package diagram start: A5> | <system diagram start: A6> | <system type diagram start: A7> | <block diagram start: A8> | <block type diagram start: A9> | <substructure 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:

/* CIF SpecificationArea */
{ <page declaration: B2> }+

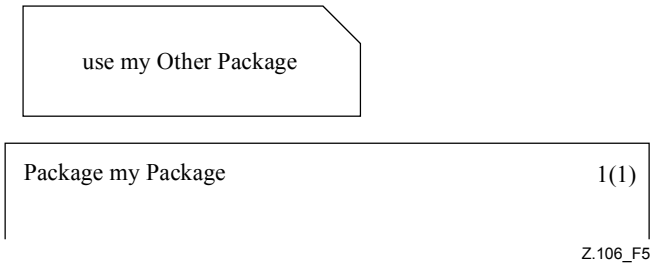
Additional information:

There should be one <page declaration: B2> for each page in the diagram.

7.4.5 A5 package diagram start:

/* 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/Z.106

7.4.6 A6 system diagram start:

/* CIF SystemDiagram */
{ <diagram parts: B1> }+
{ <package use clause> }*
<system heading**>** <end>

7.4.7 A7 system type diagram start:

/* CIF SystemTypeDiagram */
<diagram parts: B1>
<package use clause>* <system type heading> <end>

7.4.8 A8 block diagram start:

/* CIF BlockDiagram */<diagram parts: B1>
{<package use clause>}* <block heading>

7.4.9 A9 block type diagram start:

/* CIF BlockTypeDiagram */
<diagram parts: B1>
<package use clause>* <block type heading> <end>

7.4.10 A10 process diagram start:

/* CIF ProcessDiagram */
<diagram parts: B1>
{<package use clause>}* <process heading> <end>

7.4.11 A11 process type diagram start:

/* CIF ProcessTypeDiagram */
<diagram parts: B2>
<package use clause>* <process type heading> <end>

7.4.12 A12 state diagram start:

/* CIF StateDiagram */
<diagram parts: B1>
{<package use clause>}* <composite state heading> <end>

7.4.13 A13 state type diagram start:

/* CIF StateTypeDiagram */:
<diagram parts: B1>
{<package use clause>}* <composite state type heading> <end>

7.4.14 A14 state aggregation diagram start:

/* CIF StateAggregationDiagram */
<diagram parts: B1>
{<package use clause>}* <state aggregation heading> <end> **substructure**

7.4.15 A15 state aggregation type diagram start:

/* CIF StateAggregationTypeDiagram */
<diagram parts: B1>
{<package use clause>}* <state aggregation type heading> <end> **substructure**

7.4.16 A16 procedure diagram start:

/* CIF ProcedureDiagram */
<diagram parts: B1>
{<package use clause>}* <procedure heading> <end>

7.4.17 A17 operator diagram start:

/* CIF OperatorDiagram */
<diagram parts: B1>
{<package use clause>}* <operation heading> <end>

7.4.18 A18 diagram end:

{ **/* 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 CIF descriptor:

<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> | <agent reference in specification area: A37> | <package symbol: A38> | <package reference in specification area: A39> | <operator symbol: A40> | <start symbol: A41> | <stop symbol: A42> | <state symbol: A43> | <nextstate symbol: A44> | <exception handler symbol: A45> | <on exception symbol: A46> | <handle symbol: A47> | <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> | <raise symbol: A57> | <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> | <association: A72> | <specialization line: A73> | <dependency line: A74>

Additional information:

<diagram description: A2> is used to describe nested diagrams.

7.4.20 A20 default size:

/* CIF DefaultSize <size point: B22> */

Additional information:

This comment may be 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:

/* CIF CurrentPage <page name> ***/**

Additional information:

This comment may be 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 must refer to a page declared in the previous diagram start CIF comment.

Everything after this 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:

/* 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>

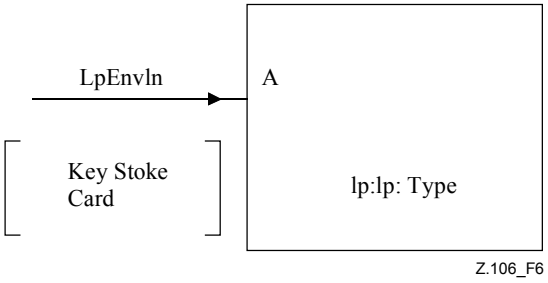
Additional information:

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/Z.106

7.4.23 A23 gate:

/* 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 definition>

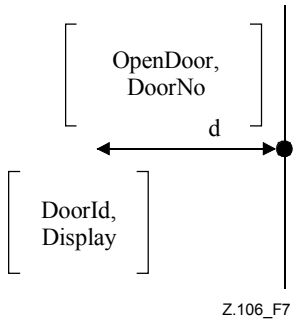
Additional information:

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 <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 <gate definition>.

The <gate constraint symbol: B3> should be given if a <textual endpoint constraint> is given.

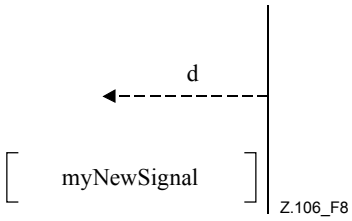
Example 1:




```
/* 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/Z.106

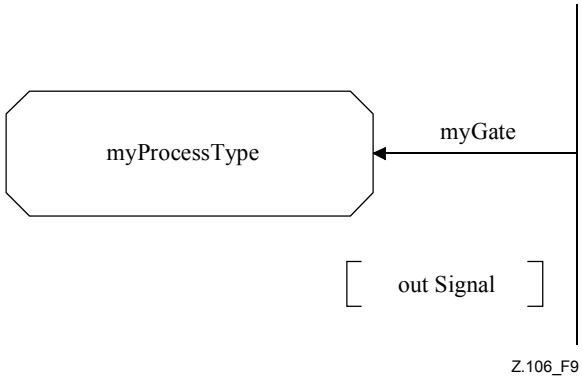
Example 2:




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

Figure 8/Z.106

Example 3:




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

Figure 9/Z.106

7.4.24 A24 gate symbol reference:

/* CIF GateSymbolReference <name> [**In** [<charstring>]] [**Out** [**<**charstring**>**]] <pointlist: B18> ***/**
<text position: B21>
[<in signal list position: B39>]
[<out signal list position: B40>]

Additional information:

This 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> defines the position of the <charstring> (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> defines the position of the <charstring> (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:

/* CIF Connect [<position: B22>]*/
[<text position: B21>]
<channel to channel connection>

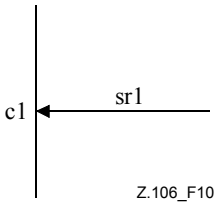
The production for <channel to channel connection> is defined in 5.6.2.

Additional information:

<position: 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/Z.106

7.4.26 A26 state connection point:

/* 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 must 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> exists, it specifies the position of the text in the <state connection point>.

7.4.27 A27 state connection:

/* CIF StateConnection <pointlist: B18> ***/**
[<text position: B21>]
<state partition connection>

The production for <state partition connection> is defined in 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> 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:

/* 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 autolayouted. 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 CIF text extension comments should not be considered as a part of the text in the symbols.

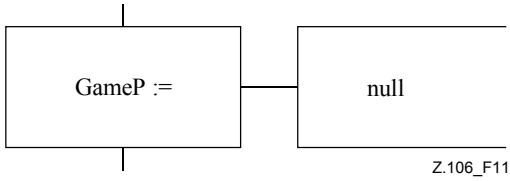
<Text extension: A28> and <Comment: A29> should be placed before the <end> in the following way: Text extensions and comments may be attached to any rule in the range <block symbol: A33> – <select symbol: A69>.

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/Z.106

7.4.29 A29 comment:

/* 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.

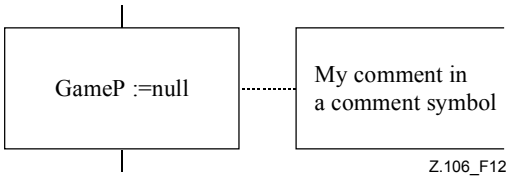
If **Left** is given, the symbol (and text) position defines the upper *right* corner.

If **Dashed** is given, the comment symbol should be drawn dashed (as a <comment symbol2> in SDL96). If **Dashed** is not given, the comment symbol should be drawn non-dashed (as a <comment symbol> in SDL92).

The <dashed line: B17> is the line connecting the comment symbol with the other symbol. If the line is not given, it will be autolayouted. 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 CIF construct is used is explained in <text extension: A28>.

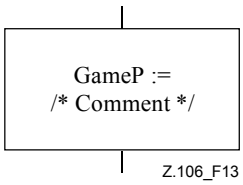
Example 1:




```
/* 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/Z.106

Related example:




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

Figure 13/Z.106

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:

/* 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:

<line: B16>

Additional information:

This CIF comment is optional. If the CIF comment is not given for a flowline, the flow line is autolayouted.

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 may be placed before or after any symbol or line SDL-CIF and SDL-PR construct within the <process body> of the diagram.

A flowline joining another flowline 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 flowlines are implicit. Flowlines after decision and transition option symbols should use <answer flow line: A32>, a 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:

/* 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 5.7.12.2.

Additional information:

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

If <line: B16> is given:

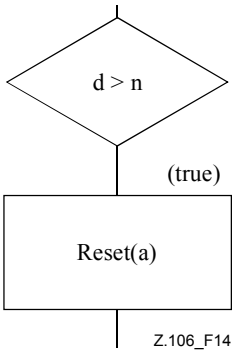
Right and **Left** have no meaning. The pointlist in the flow line specifies where the flow line starts on the decision symbol. The same rules as for <flow line: A31> apply.

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 autolayouted.

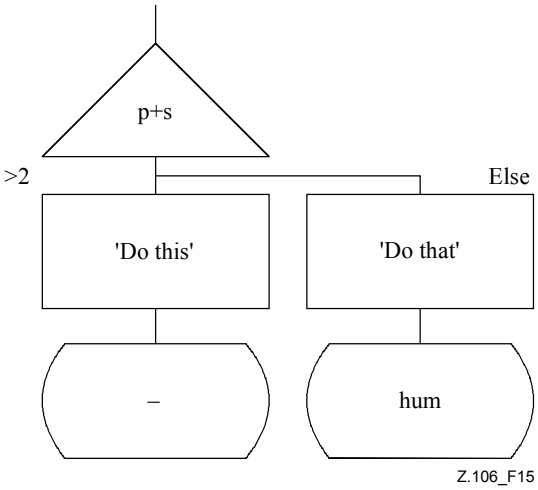
If **InvisibleBrackets** is given, the characters **(** and **)** that can be found in the SDL-PR are not visible in SDL-GR.

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/Z.106




```
/* 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/Z.106

7.4.33 A33 block symbol:

<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>s 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 autolayouted. 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:

/* CIF Block <position and size: B20> **Dashed** <block name> ***/**
[<text position: B21>]
<gate reference: B15>*

Additional information:

This comment may be placed anywhere a <textual block reference> is allowed. There should be one <dashed block symbol: A34> for each <existing typebased 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 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:

/* CIF Process <position and size: B20> **Dashed** <process name> ***/**
[<text position: B21>]
<gate reference: B15>*

Additional information:

This comment may be placed anywhere a <textual process reference> is allowed. There should be one <dashed process symbol: A36> for each <existing typebased 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 A37 agent reference in specification area:

/* CIF Agent <name> [: **<**type expression>] <position and size: B20> { **System | Block | Process** } ***/**
[<text position: B21>]

Additional information:

The symbol can only be used in specification areas.

If **System** is specified then the actual symbol shall be a <system reference area>.

If **Block** is specified then the actual symbol shall be a <block reference area>.

If **Process** is specified then the actual symbol shall be a <process reference area>.

The <text position> refers to the text corresponding to '<name> [: <type expression>]'.

Example 1:

```
/* CIF Agent P (800,550) Process */
/* CIF TextPosition (810,560) */
```

7.4.38 A38 package symbol:

/* CIF Package <position and size: B20> ***/**
[<text position: B21>]
<package reference>

Additional information:

This 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;
```

7.4.39 A39 package reference in specification area:

/* CIF PackageReference <name> <position and size: B20> ***/**
[<text position: B21>]

7.4.40 A40 operator symbol:

/* CIF Operator <identifier> <position and size: B20> ***/**
[<text position: B21>]

Additional information:

This comment may be placed anywhere an <entity in package>, <entity in system>, <entity in block>, <entity in process> or an <entity in procedure> may be placed. The operator symbol has no direct graphical connection to the <textual operator reference> (that is presented as text in a text symbol).

Example:

```
/* CIF Operator myOperator (800,550) */
```

7.4.41 A41 start symbol:

/* CIF Start <position and size: B20> ***/**
[<text position: B21>]
start [<virtuality>] [<state entry point name>] <end>

Additional information:

This 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;
```

7.4.42 A42 stop symbol:

/* CIF Stop <position and size: B20> ***/**
<stop> <end>

Example:

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

7.4.43 A43 state symbol:

/* CIF State <position and size: B20> ***/**
[<text position: B21>]
state <state list> <end> | **state** <composite state list> <end>

Examples for this CIF comment can be found in the related CIF comment <nextstate symbol: A44>.

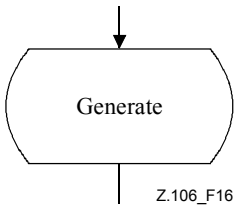
7.4.44 A44 nextstate symbol:

/* CIF NextState <position and size: B20> ***/**
[<text position: B21>]
<nextstate>

Additional information:

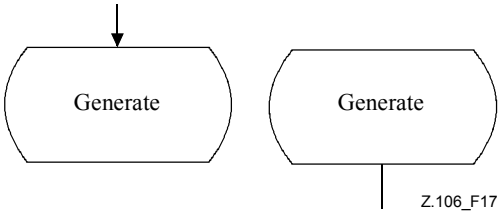
A CIF comment should be given for every state and every nextstate in SDL-PR. This means that there will be two CIF comments for one SDL-GR symbol that corresponds to both a SDL-PR state and a SDL-PR nextstate. A tool that reads a CIF file should determine if a state and a nextstate is in fact one SDL-GR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

Examples:




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

Figure 16/Z.106




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

Figure 17/Z.106

7.4.45 A45 exception handler symbol:

/* CIF Exception Handler { <position and size: B20> } ***/**
[<text position: B21>]
exceptionhandler <exception handler list> <end>

Example:

```
/* CIF Exception Handler (800,550) */
exceptionhandler myXX;
```

7.4.46 A46 on exception symbol:

/* CIF OnException <position and size: B20> ***/**
[<text position: B21>]
[<pointlist>]
onexception <exception handler name> <end>

Additional information:

The <pointlist> defines the <solid on exception association symbol> that connects the <exception handler symbol> with the other symbol. The first point in the pointlist is on the surrounding rectangle of the <exception handler symbol>. The last point in the pointlist is on the surrounding rectangle of the symbol the exception is attached to.

Note that the same graphical <exception handler symbol> can be used both as an exception handler definition and as an on-exception. This is the case if the <position and size> are the same in the Exception Handler and the OnException CIF comments.

If the <pointlist> is omitted, then autolayout shall be used for the <solid on exception association symbol>.

Example:

```
/* CIF OnException (800,550) */
onexception myXX;
```

7.4.47 A47 handle symbol:

/* CIF Handle <position and size: B20> [{ **Left** | **Right** }] ***/**
[<text position: B21>]
handle [<virtuality>] <exception stimulus 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 Handle (800,550) */
HANDLE myException;
```

7.4.48 A48 save symbol:

/* CIF Save <position and size: B20> ***/**
[<text position: B21>]
<save part>

Example:

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

7.4.49 A49 task symbol:

/* 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 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>. SDL-GR task symbols containing <export> should use <export symbol: A52>.

Example:

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

7.4.50 A50 set symbol:

/* 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 [1], 11.14).

Example:

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

7.4.51 A51 reset symbol:

/* 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 [1], 11.14).

Example:

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

7.4.52 A52 export symbol:

/* CIF Export <position and size: B20> ***/**
[<text position: B21>]
<export> <end>

Additional information:

An export symbol is a SDL-GR task symbol containing a single <export statement> (see [1], 11.14).

Example:

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

7.4.53 A53 create symbol:

/* CIF Create <position and size: B20> ***/**
[<text position: B21>]
<create request> <end>

Example:

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

7.4.54 A54 procedure call symbol:

/* CIF ProcedureCall <position and size: B20> ***/**
[<text position: B21>]
<procedure call> <end>

Example:

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

7.4.55 A55 procedure start symbol:

/* CIF ProcedureStart <position and size: B20> ***/**
[<text position: B21>]

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

Additional information:

This 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;
```

7.4.56 A56 return symbol:

/* CIF Return <position and size: B20> ***/**
[<text position: B21>]
<return> <end>

Example:

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

7.4.57 A57 raise symbol:

/* CIF Raise <position and size: B20> ***/**
[<text position: B21>]
<raise> <end>

Example:

```
/* CIF Raise (800,550) */
RAISE myException;
```

7.4.58 A58 decision symbol:

/* CIF Decision <position and size: B20> ***/**
[<text position: B21>]

DECISION <question> <end>

Additional information:

A flowline 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;
```

7.4.59 A59 continuous signal symbol:

/* 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;
```

7.4.60 A60 enabling condition symbol:

/* CIF EnablingCondition <position and size: B20> ***/**
[<text position: B21>]
<enabling condition>

Example:

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

7.4.61 A61 transition option symbol:

/* CIF TransitionOption <position and size: B20> ***/**
[<text position: B21>]

ALTERNATIVE <alternative question> <end>

Additional information:

A flowline directly after a transition option symbol should be described by an <answer flow line: A32>.

Example:

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

7.4.62 A62 join symbol:

/* 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 flowline 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;
```

7.4.63 A63 Connect:

/* CIF Connect */
[<line: B16>] [<text position: B21>]
connect [<virtuality>] [<connect list>] <end>

Additional information:

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

7.4.64 A64 label symbol:

/* 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 flowline 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 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:
```

7.4.65 A65 input symbol:

/* 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;
```

7.4.66 A66 priority input symbol:

/* 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;
```

7.4.67 A67 output symbol:

/* 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;
```

7.4.68 A68 text symbol:

/* CIF Text <position and size: B20> ***/**
[<text position: B21>]
<text>
/* CIF End Text */

Additional information:

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

A newline character before or after one of the two 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 */
```

7.4.69 A69 select symbol:

/* 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);
```

7.4.70 A70 descriptor end:

{ /* **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> |
/* **CIF End ExceptionHandler** */ **ENDEXCEPTIONHANDLER** [<exception handler name>] <end> **}**

Additional information:

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

7.4.71 A71 type reference:

/* CIF TypeReference <position and size: B20> [**Iconized**] ***/**
[<id text position: B23>] [<stereotype text position: B24>] [<icon symbol: B29>]
[<class symbol first line position: B27>] [<attribute text position: B25>]
[<class symbol second line position: B28>] [<behaviour text position: B26>]
{{**system** | **block** | **process** | **object** | **value** | **state**} **type** } | **signal** | **procedure** | **interface**} <identifier> <type reference properties>

Additional information:

If **Iconized** is part of the CIF comment, then type reference symbol should not use the class symbol but instead the special symbols for block/process/state types or procedures. Otherwise the class symbol shall be used.

If <icon symbol: B29> is present, then the class symbol shall use the small icon corresponding to the kind of type. If <icon symbol: B29> is not present, the stereotype name shall be used.

The class symbol is divided into three compartments using two lines. The top compartment of the class symbol contains the <identifier> and either the stereotype (one of **system, block, process, object, value, state, signal, procedure, interface**) or a small icon representing the stereotype. The middle compartment contains the <attribute property> part. The bottom compartment contains the <behaviour property> part.

All text and line positions are optional in this directive.

Example:

```
/*CIF TypeReference (500,400) */
/* CIF Icon */
/* CIF Line1 (500,440) */
/* CIF Line2 (500,480) */
process type P
with a integer;
with signal s ( Integer );
referenced;
```

7.4.72 A72 association:

/* CIF Association <pointlist: B18> */
[<association name text position: B30>]
[<first association end role name position: B31>]
[<first association end visibility position: B32>]
[<first association end multiplicity position: B33>]
[<first association end ordering position: B34>]
[<second association end role name position: B35>]
[<second association end visibility position: B36>]
[<second association end multiplicity position: B37>]
[<second association end ordering position: B38>]
<association definition>

Additional information:

All positions (except the <pointlist: B18>) are optional.

The first point and last points in the pointlist should be on the surrounding rectangle of the linked symbols.

Example:

```
/* CIF Association (400,300), (500,700) */
/* TextPosition (500,500) Association */
association ExampleAssociation from Pt1 from Pt2;
```

7.4.73 A73 specialization line:

/* CIF SpecializationLine <pointlist: B18> */

Additional information:

The first point in the pointlist is on the surrounding rectangle of the supertype. The last point in the pointlist is on the surrounding rectangle of the subtype.

7.4.74 A74 dependency line:

/* CIF DependencyLine <pointlist: B18> */

Additional information:

The first point in the pointlist is on the surrounding rectangle of the client (the entity that 'uses'). The last point in the pointlist is on the surrounding rectangle of the supplier (the package that 'is used').

7.5 CIF-GR Syntax – CIF B rules

The 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:

{ <page declaration: B2>+ | <nested frame: B10> }

Additional information:

If the diagram is embedded in the enclosing diagram, a <nested frame: B10> should be used.

If the diagram is not embedded in the enclosing diagram, there should be one <page declaration: B2> for each page in the diagram.

7.5.2 B2 page declaration:

/* 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:

<block symbol rectangle: B12> | <process symbol rectangle: B13>

7.5.4 B4 first signallist text position:

/* 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:

/* 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:

/* 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:

/* 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 B8 inlet text:

/* CIF InletText <macro label> ***/**
[<text position: B21>]

7.5.9 B9 outlet text:

/* CIF OutletText <macro label> ***/**
[<text position: B21>]

7.5.10 B10 nested frame:

/* CIF NestedFrame <position and size: B20> ***/**
[<diagram heading text position: B21>]

7.5.11 B11 frame declaration:

/* CIF Frame <position and size: B20> ***/**

7.5.12 B12 block symbol rectangle:

/* CIF Block <position and size: B20> ***/**

7.5.13 B13 process symbol rectangle:

/* CIF Process <position and size: B20> ***/**

7.5.14 B14 package use symbol:

/* CIF Use <position and size: B20> ***/**
[<text position: B21>]

Additional information:

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

7.5.15 B15 gate reference:

/* CIF GateReference [<name>] <connection point: B22> ***/**
<text position: B21>

Additional information:

This 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 can be found in the SDL-PR for connected channels. Read more about gates and gate references in <channel: A22>. See also the example. Use this 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) */
```

7.5.16 B16 line:

/* CIF Line <pointlist: B18> ***/**

7.5.17 B17 dashed line:

/* CIF Line <pointlist: B18> **Dashed */**

7.5.18 B18 pointlist:

<point: B22> {, <point: B22> }+

7.5.19 B19 page text position:

/* CIF TextPosition <point: B22> **PageName */**

7.5.20 B20 position and size:

<position point: B22> [, <size point: B22>]

Additional information:

The size point may be omitted 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.

7.5.21 B21 text position:

/* 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) */
```

7.5.22 B22 point:

(<integer>, <integer> **)**

7.5.23 B23 id text position:

/* 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.

7.5.24 B24 stereotype text position:

/* CIF TextPosition <point: B22> **Stereotype */**

Additional information:

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

7.5.25 B25 attribute text position:

/* CIF TextPosition <point: B22> **Attribute */**

Additional information:

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

7.5.26 B26 behavior text position:

/* CIF TextPosition <point: B22> **Behavior */**

Additional information:

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

7.5.27 B27 class symbol first line position:

/* CIF Line1 <point: B22> ***/**

Additional information:

This CIF comment is used to specify the position for a line dividing the top and middle compartments of a class symbol. The point defines the left endpoint of the line.

Example:

```
/* CIF Line1 (800,550) */
```

7.5.28 B28 class symbol second line position:

/* CIF Line2 <point: B22> ***/**

Additional information:

This CIF comment is used to specify the position for a line dividing the middle and bottom compartments of a class symbol. The point defines the left endpoint of the line.

Example:

```
/* CIF FirstLine (900,550) */
```

7.5.29 B29 icon symbol:

/* CIF Icon [<position and size: B20>] ***/**

7.5.30 B30 association name text position:

/* CIF TextPosition <point: B22> **Association */**

Additional information:

This text position defines the upper left corner of the surrounding rectangle of the text for the association name.

7.5.31 B31 first association end role name position:

/* CIF TextPosition <point: B22> **Role1 */**

Additional information:

This text position defines the upper left corner of the surrounding rectangle of the text for the role name.

7.5.32 B32 first association end visibility position:

/* CIF TextPosition <point: B22> **Visibility1 */**

Additional information:

This text position defines the upper left corner of the surrounding rectangle of the text for the visibility text.

7.5.33 B33 first association end multiplicity position:

/* CIF TextPosition <point: B22> **Multiplicity1 */**

Additional information:

This text position defines the upper left corner of the surrounding rectangle of the text for the multiplicity text.

7.5.34 B34 first association end ordering position:

/* CIF TextPosition <point: B22> **Ordering1 */**

Additional information:

This text position defines the upper left corner of the surrounding rectangle of the text for the ordering text.

7.5.35 B35 second association end role name position:

/* CIF TextPosition <point: B22> **Role2 */**

Additional information:

This text position defines the upper left corner of the surrounding rectangle of the text for the role name.

7.5.36 B36 second association end visibility position:

/* CIF TextPosition <point: B22> **Visibility2 */**

Additional information:

This text position defines the upper left corner of the surrounding rectangle of the text for the visibility text.

7.5.37 B37 second association end multiplicity position:

/* CIF TextPosition <point: B22> **Multiplicity2 */**

Additional information:

This text position defines the upper left corner of the surrounding rectangle of the text for the multiplicity text.

7.5.38 B38 second association end ordering position:

/* CIF TextPosition <point: B22> **Ordering2 */**

Additional information:

This text position defines the upper left corner of the surrounding rectangle of the text for the ordering text.

7.5.39 B39 in signal list position:

/* 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.

7.5.40 B40 out signal list position:

/* 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 CIF comments

7.6.1 C0 tool-specific CIF comment:

/* CIF [**Keep**] **Specific** <name of tool> <tool-specific information> ***/**

Additional information:

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

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

Informal example (not correct 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';
```

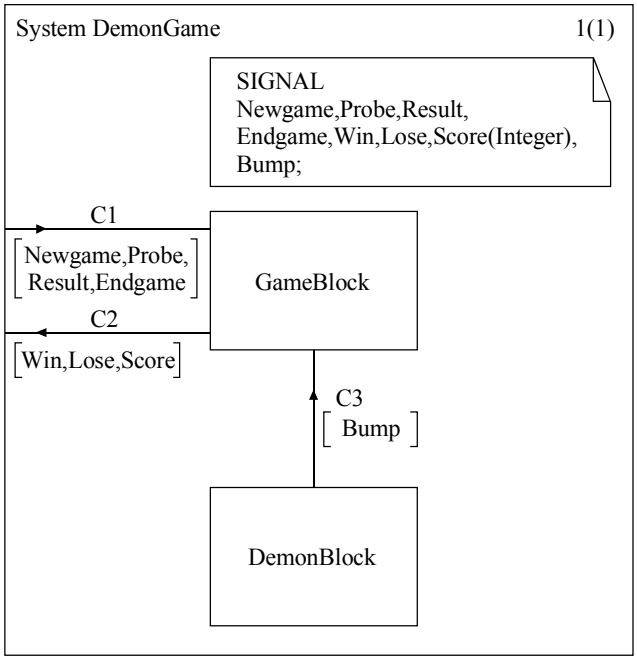
8 Examples

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 CIF to express nested diagrams, where the SDL-PR does not have the keyword REFERENCED.

8.1 DemonGame

8.1.1 System DemonGame

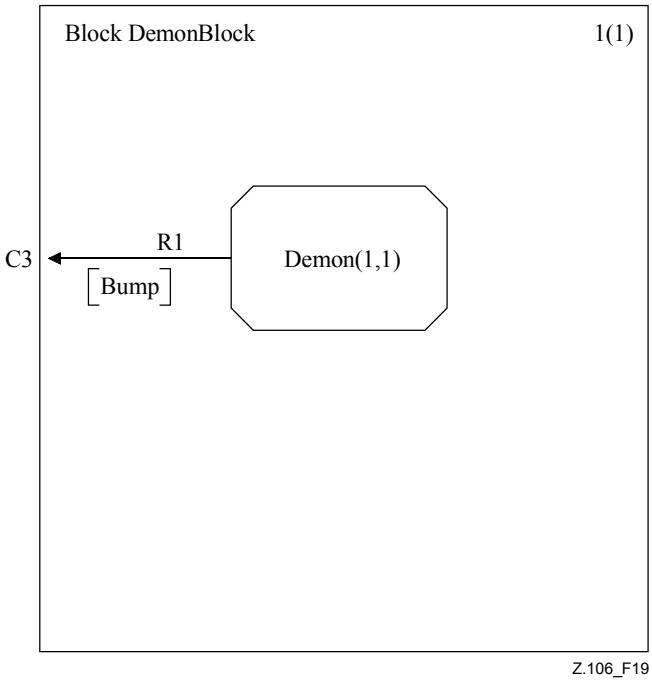



```
/* CIF SystemDiagram */
/* CIF Page 1 (1150,1000) */
System DemonGame;
/* CIF Specific mySDLTool Page 1
NoGrid */
/* CIF DefaultSize (300,200) */
/* CIF Text (400,100),(600,100) */
SIGNAL
Newgame,Probe,Result,
Endgame,Win,Lose,Score(Integer),Bump;
/* CIF End Text */
/* CIF Channel (550,700),(550,500) */
/* 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/Z.106 – System DemonGame

8.1.2 Block DemonBlock

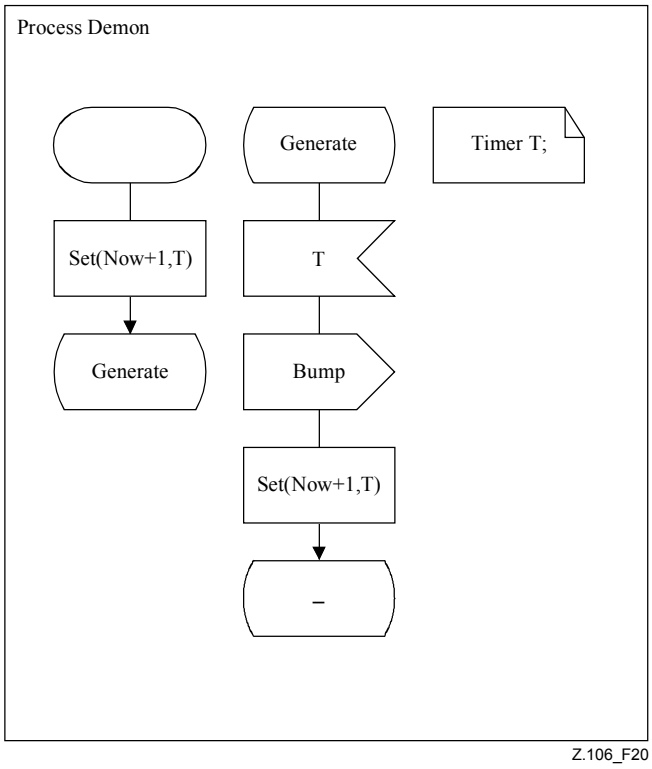



```
/* CIF BlockDiagram */
/* CIF Page 1 (1000,1000) */
/* CIF Frame (100,100),(800,800)
*/
Block DemonBlock;
/* CIF Specific mySDLTool Page 1
NoGrid */
/* CIF DefaultSize (300,200) */
/* CIF Connect */
/* CIF TextPosition (25,300) */
Connect C3 and R1;
/* CIF SignalRoute
(250,350),(0,350) */
/* CIF 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/Z.106 – 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;
```

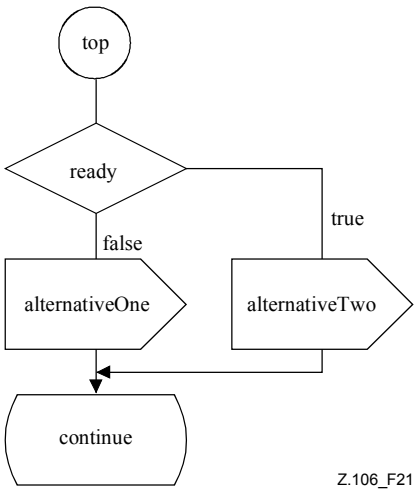
SDL-GR **SDL-CIF**

Figure 20/Z.106 – Process Demon

8.2 Tricky SDL constructs

8.2.1 Joining flowlines 1

Note that the pointlist for the flowline 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/Z.106 – Joining flowlines

8.2.2 Joining flowlines 2

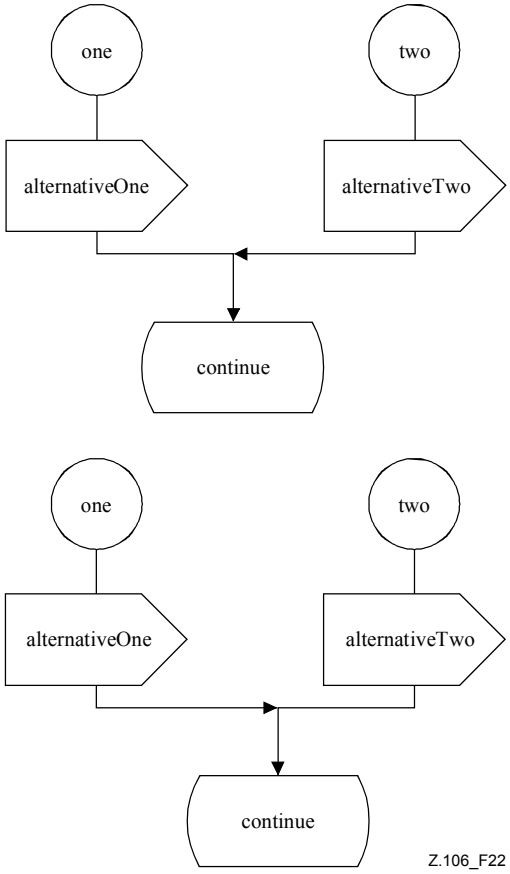
This example is very similar to the previous one, in fact it is the same SDL-GR. The SDL-PR is expressed differently though. Note that the two joining flowlines are placed close to the symbol they are coming from in the SDL-PR below.

```
/* 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 flowlines 3

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

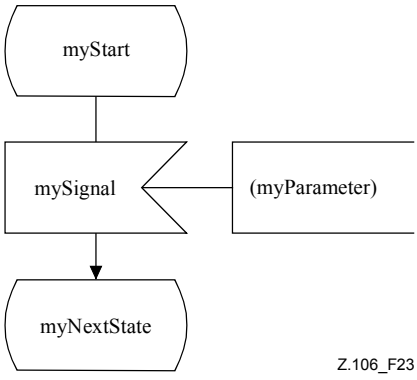



```
/* CIF Label (150,100),(100,100)
*/
connection
  one:
  /* CIF Output (100,300) */
  output alternativeOne;
  /* CIF Label Invisible */
  grs0:
  /* CIF NextState (250,500) */
  nextstate continue;
/* CIF End Label */
endconnection one;
/* CIF Label (450,100),(100,100)
*/
connection
  two:
  /* CIF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

Figure 22/Z.106 – Joining flowlines

8.2.4 Lines and enclosing rectangles

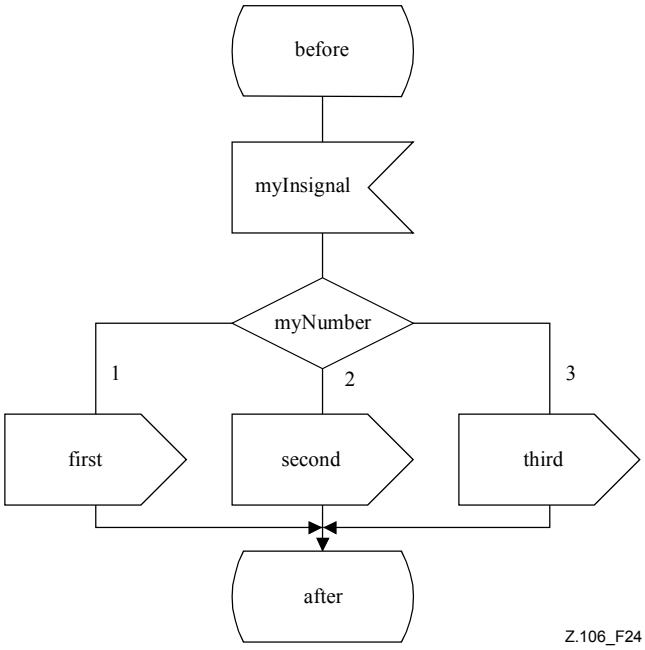
Note that the text extension line is described as if its endpoints were on the border of the surrounding rectangles of the attached symbols, even if this is not completely true in SDL-GR. (The endpoint on the input symbol is inside the surrounding rectangle.) This rule applies to all lines/channels connected 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/Z.106 – Lines and enclosing rectangles

8.2.5 Answer flow lines after decision

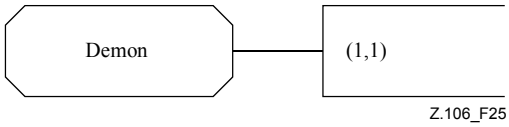



```
/* CIF State (400,100) */
state before;
  /* CIF Input (400,300) */
  input myInsignal;
    /* CIF Decision (400,500) */
    decision myNumber;
    /* CIF Answer Left
InvisibleBrackets */
    (1):
      /* CIF Output (100,700) */
      output First;
    /* CIF Answer
InvisibleBrackets */
    (2):
      /* 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/Z.106 – Answer flow lines

8.2.6 Text extension

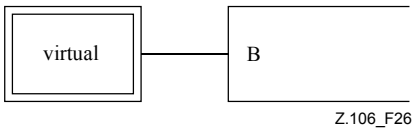
This is allowed:




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

Figure 25/Z.106 – Text extension

This is also allowed:




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

Figure 26/Z.106 – Text extension

8.2.7 Nested diagrams

This nested diagram is converted into two diagrams with the following steps:

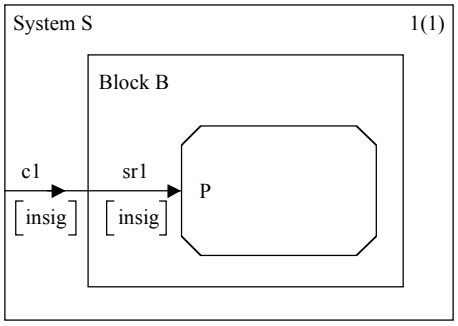
1) Replace the block diagram in system S with a block reference symbol.

2) Create a separate block diagram.

3) Insert information about what external channel the internal channel is connected to.

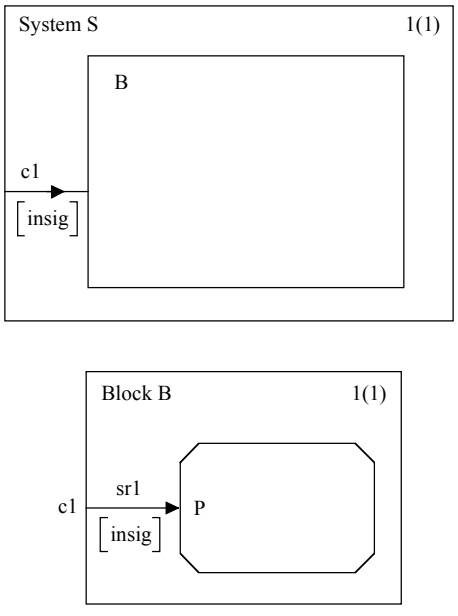
4) Name the page with an appropriate name.

a) *Nested form*




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

b) *After conversion/without nesting*



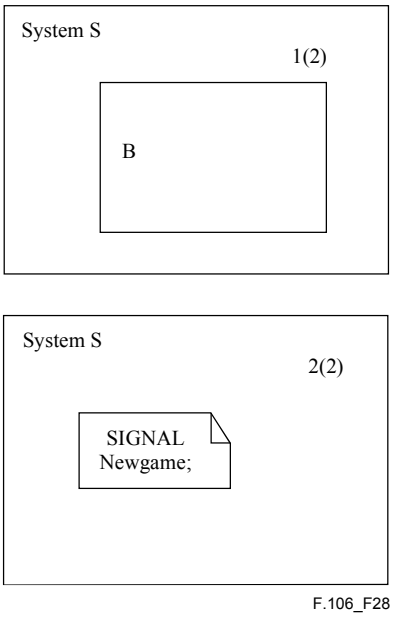
Z.106_F27

```
/* 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;
```

Figure 27/Z.106 – Nested diagrams

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/Z.106

8.3 Situations CIF cannot handle

Figure 29/Z.106

CIF cannot handle this properly. The CIF code would be:

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

The comment in the SDL-GR is lost. The consequence is that CIF cannot be used 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 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 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 you express this in SDL-PR?

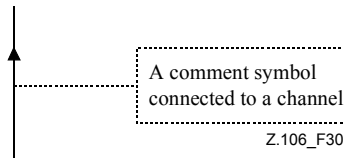


Figure 30/Z.106

Here is one more SDL-GR example, that may or may not be legal SDL-GR, that neither 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 we are converting back to SDL-GR?)

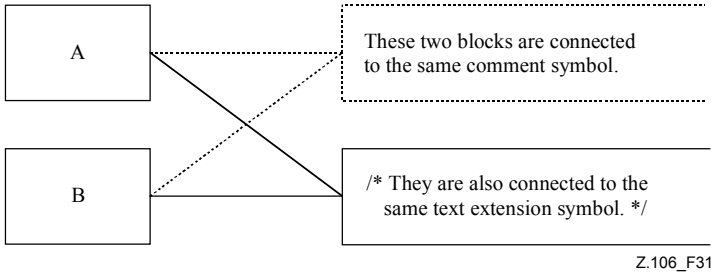


Figure 31/Z.106

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

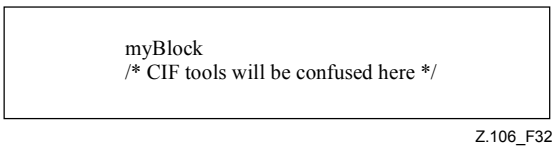


Figure 32/Z.106

9 CIF conformance criteria

9.1 About tools reading a CIF file

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

Example 1: The CIF file says 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 CIF comment.

Example 2: The 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 CIF comment.

9.2 Automatic vs. forced layout

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

9.3 Retainment and use of tool-specific information

The CIF has been intentionally defined to provide tool-specific information that may or may not be used by another tool. ITU-T recommends that, where tool vendors use tool-specific information, they provide this information publicly to avoid a clash with other tools. An example has been provided in this Recommendation and future CIF versions may 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 CIF comments

I.1 Maintenance of CIF

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

First of all, names of new tools must be approved by the ITU-T Z.106 Working Group, to ensure that there is no name conflict.

In the perspective of enriching SDL-CIF, tool makers who have created new tool-specific directives should propose them to the Z.106 Working Group. 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 Z.106 Working Group, in order to adjust to mandatory/optional classification of directives.

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

I.2 Current tool-specific CIF comments

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

How a tool should handle tool-specific CIF comments is defined in the subclause about tool-specific CIF comments in this Recommendation.

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

I.2.1 Placement of tool-specific CIF comments

This topic is discussed in the additional information associated with <tool-specific CIF comments: C0>.

I.2.2 Example

In the example below, many tool-specific CIF comments are associated with the CIF rule <system diagram: A6>. Detailed information for each tool-specific 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:

/* CIF Specific mySDLTool Version x.y */
<diagram description: A2>

Association:

This tool-specific CIF comment is associated with an A rule mentioned in the rule for <diagram start: A3>.

Example (where this tool-specific 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:

/* CIF Specific mySDLTool OriginalFileName <file name char string literal> ***/**
<diagram start: A3>

Association:

This tool-specific 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 CIF. When the 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 char string literal>, not the complete path (/home/lat/a.ssy).

Example (where this tool-specific 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:

/* CIF Specific mySDLTool Page <page number> [**ShowMeFirst**] [**Scale** <integer>] [**Grid** <point: B22>] [**AutoNumbered**] [**FixedHeadingSize** [<fixed size point: B22>]] ***/**

Association:

This tool-specific CIF comment is associated with an A rule mentioned in the rule for <diagram start: A3>. There may be zero or one of this tool-specific CIF comment for each <page declaration: B2> in the diagram.

Additional information:

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

– If 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.).

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

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

– If 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 charstring literal> must 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 CIF comments.

In addition, if FixedHeadingSize is given, this tool-specific CIF comment says 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 the symbol to the larger size. The SDL tool uses this comment for additional heading symbols that the user has "collapsed" with a double click.

Example (where this tool-specific 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:

/* CIF Specific mySDLTool FixedSize [<fixed size point: B22>] ***/**

Association:

This tool-specific CIF comment is associated with <text symbol: A68>.

Additional information:

This tool-specific CIF comment says that although the symbol we have just defined in a CIF comment was given a size (a size that makes the complete text visible), we use this fixed smaller size 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 the symbol to the larger size. The SDL 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 B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	**Languages and general software aspects for telecommunication systems**

Geneva, 2003