INTERNATIONAL TELECOMMUNICATION UNION

ITU-T Z.104

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(10/2004)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

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

Encoding of SDL data

ITU-T Recommendation Z.104

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
Testing and Test Control 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.104

Encoding of SDL data

Summary

Encoding of SDL data enables communication of data values between pieces of SDL in an implementation independent way.

Data values can be encoded in a text format defined by this Recommendation. Alternatively, when the data is defined in ASN.1, sets of standardized ASN.1 encoding rules can be invoked.

The results of encoding can be used internally in the SDL model or to communicate between SDL models or between SDL and other non-SDL elements such as test environments.

Source

ITU-T Recommendation Z.104 was approved on 7 October 2004 by ITU-T Study Group 17 (2001-2004) under the ITU-T Recommendation A.8 procedure.

Keywords

ASN.1, Data, Encoding, ITU-T Rec. Z.100, ITU-T Rec. Z.105, SDL.

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.

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

INTELLECTUAL PROPERTY RIGHTS

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

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

Introduction

Encoding provides a way to specify how data values are encoded when information is communicated between pieces of SDL.

In general, only values can be passed between separate pieces of SDL and between SDL and its environment. For this reason, encoding as defined by this Recommendation only applies to value sorts of data: encoding of object sorts of data is not supported. This is consistent with the use of data within SDL signal communication, where replicates of the object data items are created (except in the case that an instance of a process contains both the sender and receiver).

It is expected that the specification of encoding for SDL data will normally be used either on a channel that represents a normative interface, or for data to be encapsulated in some other data item to be handled in some transparent way.

When encoding is specified on a channel, this will be transparent to the rest of the SDL model if the receiving agents do not use the additional syntax for input given in the Recommendation. If neither end of the channel lead to the SDL environment, the encoding specification is merely adding the encoding requirements to the rest of the SDL and the encoding is removed during normal signal input. If the source or destination of the signal is the SDL environment, the encoding is putting a requirement on the encoding of messages to or from the environment.

The encapsulation of data within another data item typically happens within layered protocols where communication between the two layers can correspond to a channel between two SDL agents. Encoding provides the possibility to simplify SDL models. At one layer the signal output can be encoded, but when signal is received from the channel, the encoding can be ignored so that a signal of any of the encoded types is received. The data can be stored at the receiver layer (or be encapsulated for further layers) and be passed to a peer. When the data is output to the original layer, decoding can take place, restoring the original signal.

Encoding and decoding are also provided as data expressions so that encapsulation (and its reversal) does not have to be done as part of input and output.

Other uses of encoding are envisaged, particularly for the text encoding. The text encoding is likely to be useful for testing and validating SDL models, as well as providing an ASN.1 independent encoding, though the set of text encoding rules is designed for transfer of information as text characters to other automata rather than to be read by humans.

Because encoding is primarily concerned with communication, encoding is related to the data conveyed by sets of signals and is related to interface, gates and channels. There is specific support for the use of an ASN.1 ABSTRACT-SYNTAX name in an interface to imply a set of signals based on a top level CHOICE type in the ABSTRACT-SYNTAX.

This Recommendation does not provide for the specification of encoding for variables.

ITU-T Recommendation Z.104

Encoding of SDL data

1 Scope

This Recommendation extends SDL to provide mechanisms to specify data encoding, so that data passed between different parts of an SDL system (or between SDL systems) has an implementation-independent encoding.

A mechanism is provided that allows the encoding to be based on different rules. One encoding is defined where the result is a text string. Alternatively, for data that is based on ASN.1 definitions or can be mapped to ASN.1 definitions, the sets of encoding rules defined in the X.69x series of Recommendations can be used.

1.1 Use of encode and decode

Within a separately implemented piece of SDL, the way data values are represented does not need to be known and the implementation ensures that the data behaves in the way expected.

When information is communicated between pieces of SDL that are separately implemented, data values need to be passed using an encoding that can be processed by all the pieces of SDL that use the information. For example, if a Boolean value is passed, it could be passed as a single bit and it would need to be defined if a zero bit represents true or false. To enable the Boolean value to be communicated correctly, both sender and receiver need to apply the same set of encoding rules. Before the value is sent, or once the value has been received, it is possible that the information may be represented in a different way. For example, in part of a system with good support for 16-bit Integers, where the amount of storage is not important but accessing individual bits is inefficient, a Boolean might be best represented 16 bits, with 16 zero bits representing false and any other set of bits representing true. In another part of the same system Boolean may be better implemented as a single octet.

Communication in SDL takes place via channels, which are explicit or implicit. Channel communication can be between parts of the SDL system, or between parts of the system and the environment.

The communication that takes place on a channel either corresponds to a protocol or can be considered as a protocol. In general, a channel used for communication carries several different signals in each direction. Typically, the signals in one direction are different from the signals in the other direction. Occasionally, a channel conveys signals in one direction only, but this is unusual.

A channel between two agents carries the signals for a protocol between those two agents. If it assumed that any type of signal can be received at any time, each encoded type of signal in one direction must be distinct from all other types of signals in the same direction, otherwise, the receiving agent will not be able to distinguish one signal from another. Two instances of a signal with the same data output from the same agent on the same channel will have the same encoding. If the type of signal or the data or the originating agent instance is different, the encoded information is different. The originating agent is included because every signal instance conveys the pid of its originating agent instance as an implicit parameter, otherwise two signal instances of a signal with the same data output on the same channel have the same encoding.

2 References

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

– ITU-T Rec. X.680 (2002) | ISO/IEC 8824-1:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.

– ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*.

– ITU-T Rec. X.691 (2002) | ISO/IEC 8825-2:2002, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*.

– ITU-T Rec. X.693 (2001) | ISO/IEC 8825-4:2002, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)*.

– ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*.

– ITU-T Rec. Z.100 (2002), *Specification and Description Language (SDL)*.

– ITU-T Rec. Z.105 (2003), *SDL combined with ASN.1 modules (SDL/ASN.1)*.

3 Definitions

3.1 **decode** (process): The process to construct an SDL data value from a text string or bit-pattern that is assumed to be an encoding of the SDL data value using the same encoding rules as those used in the decode.

3.2 **decoding**: The result of a decode process.

3.3 **encode** (process): The process of producing an encoding.

3.4 **encoding**: The text string or bit-pattern resulting from the application of a set of encoding rules to an SDL data value.

3.5 **set of encoding rules**: One of the sets of (ASN.1) encoding rules defined in the X.69x series of ITU-T Recommendations, or the set of text encoding rules defined in 10.7.1 and Annex A of this Recommendation, or an implementation-dependant or application-dependant set of encoding rules defined by a procedure code invoked according to this Recommendation (see 10.7).

4 Abbreviations

ASN.1 Abstract Syntax Notation One

SDL Specification and Description Language

5 Conventions

The notation and presentation conventions of ITU-T Rec. Z.100 are used in this Recommendation.

This Recommendation is organized so that numbering of clauses 6 to 13 follows the numbering in ITU-T Rec. Z.100.

Where an abstract or concrete syntax rule is defined in this Recommendation with the same name as a rule in ITU-T Rec. Z.100, the rule given here replaces the rule in ITU-T Rec. Z.100. Any *Grammar* conditions, *Semantics* and *Model* defined on a named rule in ITU-T Rec. Z.100 apply to the redefined rule, unless specifically defined otherwise in this Recommendation.

Characters are identified by the names (in uppercase letters) they are given in ISO/IEC 10646.

6 General rules

This Recommendation adds additional keywords **encode** and **decode** to the lexical rule <keyword>.

6.1 Lexical rules

<keyword> ::=

abstract	**active**	**adding**
| **aggregation**	| **alternative**	| **and**
| **any**	| **as**	| **association**
| **atleast**	| **block**	| **break**
| **call**	| **channel**	| **choice**
| **comment**	| **composition**	| **connect**
| **connection**	| **constants**	| **continue**
| **create**	| **dcl**	| **decision**
| **decode**	| **default**	| **else**
| **encode**	| **endalternative**	| **endblock**
| **endchannel**	| **endconnection**	| **enddecision**
| **endexceptionhandler**	| **endinterface**	| **endmacro**
| **endmethod**	| **endobject**	| **endoperator**
| **endpackage**	| **endprocedure**	| **endprocess**
| **endselect**	| **endstate**	| **endsubstructure**
| **endsyntype**	| **endsystem**	| **endtype**
| **endvalue**	| **env**	| **exception**
| **exceptionhandler**	| **export**	| **exported**
| **external**	| **fi**	| **finalized**
| **from**	| **gate**	| **handle**
| **if**	| **import**	| **in**
| **inherits**	| **input**	| **interface**
| **join**	| **literals**	| **loop**
| **macro**	| **macrodefinition**	| **macroid**
| **method**	| **methods**	| **mod**
| **nameclass**	| **nextstate**	| **nodelay**
| **none**	| **not**	| **now**
| **object**	| **offspring**	| **onexception**
| **operator**	| **operators**	| **optional**
| **or**	| **ordered**	| **out**
| **output**	| **package**	| **parent**
| **priority**	| **private**	| **procedure**
| **protected**	| **process**	| **provided**
| **public**	| **raise**	| **redefined**
| **referenced**	| **rem**	| **remote**
| **reset**	| **return**	| **save**
| **select**	| **self**	| **sender**
| **set**	| **signal**	| **signallist**
| **signalset**	| **size**	| **spelling**
| **start**	| **state**	| **stop**
| **struct**	| **substructure**	| **synonym**
| **syntype**	| **system**	| **task**
| **then**	| **this**	| **timer**
| **to**	| **try**	| **type**
| **use**	| **value**	| **via**
| **virtual**	| **with**	| **xor**

7 Organization of SDL specifications

7.1 Framework

The Framework of SDL specifications is as defined in ITU-T Rec. Z.100.

7.2 Package

Model

When an ASN.1 module is used as a package and a <definition selection> in the <package use clause> referring to the ASN.1 has the <selected entity kind> **interface** and the <name> is the name of a CHOICE data type, there is an implied interface. This implied interface has the same name as the CHOICE and defines signals equivalent to the CHOICE alternatives.

8 Structural Concepts

The structural concepts are as defined in ITU-T Rec. Z.100 with the addition of the optional identification of a set of encoding rules for gates and channels. Otherwise, the structural concepts are as defined in ITU-T Rec. Z.100.

8.1 Types, instances and gates

This Recommendation adds the optional identification of a set of encoding rules to the definition of gates.

8.1.1 Structural type definitions

The structural type definitions are as defined in ITU-T Rec. Z.100.

8.1.2 Type expressions

Type expressions are as defined in ITU-T Rec. Z.100.

8.1.3 Definitions based on types

Definitions based on types are as defined in ITU-T Rec. Z.100.

8.1.4 Abstract types

Abstract types are as defined in ITU-T Rec. Z.100.

8.1.5 Gates

A gate can have a set of encoding rules. An output of signal from an agent via the gate is encoded as specified by the set of encoding rules. Information received via the gate is decoded according to the set of encoding rules. If no specific set of encoding rules is given, the encoding is not defined by the SDL specification.

Abstract grammar

Gate-definition	::	*Gate-name* [*Encoding-rules*] *In-signal-identifier-**set*** *Out-signal-identifier-**set***

If there is a set of *Encoding-rules*, the *In-signal-identifier-**set*** and *Out-signal-identifier-**set*** shall not contain any implicit signals for remote procedures or remote variables.

If an external channel with a set of *Encoding-rules* is connected to the gate of an agent or composite state, the *Encoding-rules* of the *Gate-definition* for the gate shall be the same as the *Encoding-rules* for the channel. There shall be, at most, one such channel connected to the gate and the signals conveyed in a direction for the gate shall be the same as the signals conveyed in the corresponding direction in the channel.

If an internal channel of an agent or agent type is connected to the gate of the agent or agent type that has a *Gate-definition* with a set of *Encoding-rules*, the *Encoding-rules* of the channel shall be the same. The signals conveyed in a direction for the gate shall be the same as the signals conveyed in the corresponding direction in the channel. There can be more than one such internal channel.

Concrete grammar

<gate definition> ::=
{ <gate symbol> | <inherited gate symbol> }
is associated with { <gate> [<encoding rules>]
[<signal list area>] [<signal list area>] } ***set***
[***is connected to*** <endpoint constraint>]

NOTE 1 – When a gate in a subtype is an extension of a gate inherited from a supertype, the <inherited gate symbol> is used in the concrete syntax.

NOTE 2 – If a different encoding is required in each direction, the communication has to be specified by two gates that each carry signals in one direction.

<interface gate definition> ::=
<gate symbol 1>
is associated with { <interface identifier> [<encoding rules>] }

A specification of <encoding rules> that is associated with an <inherited gate symbol> shall specify the same set of *Encoding-rules* as the *Encoding-rules* of the corresponding gate definition in the supertype if that gate has an *Encoding-rules*. If there is no set of <encoding rules> that is associated with an <inherited gate symbol>, and there is a set of *Encoding-rules* for the gate in the supertype, the inherited gate has this set of *Encoding-rules*. If there is no set of *Encoding-rules* for the gate in the supertype, the presence and value of the inherited gate *Encoding-rules* is determined by the presence and value of the <encoding rules>.

Semantics

The set of *Encoding-rules* of a *Gate-definition* of an agent type or composite state type corresponds to the set of *Encoding-rules* of the channel in the enclosing scope in (the set of) instance specifications. The encoding used in the behaviour of the type can be determined independently of the actual channel that is connected to the gate of the instance.

Model

The set of *Encoding-rules* of a *Gate-definition* of the implied agent type of an <agent diagram> (or the implied composite state type of a <composite state area>) is the same as the set of *Encoding-rules* of the external channel (explicit or implied) from which the *Gate-definition* is derived. There is no *Encoding-rules* in this *Gate-definition* if there is none in the external channel.

The set of *Encoding-rules* of an implied *Gate-definition* of a system *Agent-type-definition* (whether defined by a <system type> or implied from a <system diagram>) is the same as the set of *Encoding-rules* of the internal channel (explicit or implied) from which the *Gate-definition* is derived. There is no *Encoding-rules* in this *Gate-definition* if there is none in the internal channel.

If an explicit <gate on diagram> is given for the <system type diagram> or <system diagram>, the set of *Encoding-rules* of the corresponding *Gate-definition* of the *Agent-type-definition* is determined by the <encoding rules> of the <gate on diagram> if the <encoding rules> is present. If the <encoding rules> is absent, the set of *Encoding-rules* is determined from the internal channel in the same way as for an implied *Gate-definition*.

8.2 Context parameters

Context parameters (including gate context parameters) are as defined in ITU-T Rec. Z.100.

NOTE – When a gate in parameterized type is defined by a formal context parameter, the set of encoding rules of the gate in a specialized type used to define instances will be determined by the actual gate definition identified by the actual context parameter.

8.3 Specialization

Specialization is as defined in ITU-T Rec. Z.100.

8.4 Type references

Type references are as defined in ITU-T Rec. Z.100.

NOTE – A <gate property area> in a type reference is a <gate definition> or <interface gate definition> and therefore can contain an <encoding rules> specification which must be consistent with the definition given with the type.

8.5 Associations

Associations are as defined in ITU-T Rec. Z.100.

9 Agents

Agents are as defined in ITU-T Rec. Z.100.

10 Communication

Encoding of data extends the definition of channels and connections.

The encoding rules grammar is defined.

10.1 Channel

A channel connecting two agents determines the encoding (if any) to be used for communication between the agents. A channel that is connected to the environment of an agent, has the encoding defined (if any) for that gate connecting it with the environment.

Abstract grammar

Channel-definition :: *Channel-name*
[*Encoding-rules*]
[**NODELAY**]
*Channel-path-**set***

The *Originating-gate* or *Destination-gate* shall have the same *Encoding-rules* as the *Channel-definition*. If the *Channel-definition* has no *Encoding-rules*, neither the *Originating-gate* nor *Destination-gate* shall have an *Encoding-rules*.

Concrete grammar

<channel definition area> ::=
<channel symbol>
is associated with
{ [<channel name> [<encoding rules>]]
{ [<signal list area>] [<signal list area>] } ***set*** }
is connected to {
{ <agent area> | <state partition area> | <gate on diagram> }
{ <agent area> | <state partition area> | <gate on diagram> } } ***set***

NOTE 1 – If a different encoding is required in each direction, the communication has to be specified by two channels that each carry signals in one direction.

Semantics

The *Encoding-rules* of a *Channel-definition* connected to a set of instance specifications is used in the behaviour of the instances.

Model

If the <encoding rules> is omitted and the <channel symbol> is connected to a <gate on diagram> with an <encoding rules>, the *Channel-definition* has the same *Encoding-rules* as the *Gate-definition* of the <gate on diagram>. If the *Gate-definition* has no *Encoding-rules,* the *Channel-definition* has no *Encoding-rules*.

Implicit channels have no *Encoding-rules* if the gates they are connected to have no *Encoding-rules*. Otherwise the *Encoding-rules* of an implicit channel is the same as the *Encoding-rules* of each of the gates.

NOTE 2 – If the channel is connected to the frame of the enclosing diagram and there is no <gate on diagram>, this represents a connection.

10.2 Connection

Model

The *Encoding-rules* of an implicit gate of a connection is the same as the connected external channel. A channel without an <encoding rules> (or an implicit channel) that is connected to a gate that has an *Encoding-rules* has the same *Encoding-rules*.

10.3 Signal

Signals are as defined in ITU-T Rec. Z.100.

10.4 Signal list definition

Signal list definitions are as defined in ITU-T Rec. Z.100.

10.5 Remote procedures

Remote procedures are as defined in ITU-T Rec. Z.100.

10.6 Remote variables

Remote variables are as defined in ITU-T Rec. Z.100.

10.7 Communication path encoding rules, encode and decode

The set of encoding rules specifies which set of encoding rules is used to encode and decode data conveyed by a particular channel or gate.

Abstract grammar

Encoding-rules	::	*Rules-identifier*
Encoding-expression	::	*Signal-identifier* [*Expression*]* *Encoding-path*
Encoding-path	::	{ *Channel-identifier* | *Gate-identifier* }
Decoding-expression	::	*Expression* *Encoding-path*
Rule-identifier	::	*Literal-identifier*

The *Rule-identifier* shall be one of the literal identifiers of the data type `Encoding` (see below in *Semantics*). If the actual rule identified corresponds to an encoding defined in the X.69x series of ITU-T Recommendations, the set of signals carried by an *Encoding-path* shall correspond to elements of an ASN.1 CHOICE type that is carried by the channel. The *Encoding-path* and ASN.1 CHOICE type correspond in one direction if each signal name corresponds to a CHOICE name and for each signal the (single) parameter of the signal is the same as the data type of the corresponding CHOICE.

The data type `Encoding` shall be the Predefined data type `Encoding` or a data type with the name `Encoding` that is a specialization (direct or indirect) of the Predefined data type `Encoding`. The specialization shall only add literal names to the Predefined data type `Encoding` and shall not change any other properties.

If the *Rule-identifier* corresponds (by *Name*) to an `Encoding` literal defined in the **`package`** `Predefined`, built-in procedures implied by the standardized sets of encoding rules are invoked (and other procedures – even if visible with names corresponding as below – are ignored).

If the *Rule-identifier* corresponds to an additional literal added to a specialization of the predefined data type `Encoding`, there shall be a visible procedure with the appropriate signature for each invocation (implicit or explicit) of the *Encoding-expression* or *Decoding-expression*.

The name of the procedure for encode is the name `encode` concatenated with the name of an additional `Encoding` literal (for example, `encodemyprotocol` where the additional `Encoding` literal is `myprotocol`). This procedure shall have one parameter of the implicit choice type for the relevant path for the invocation (see below in *Semantics*). The procedure shall return a Charstring, Bitstring or Octetstring.

The name of the procedure for decode is the name `decode` concatenated with the name of an additional `Encoding` literal (for example, `decodemyprotocol` where the additional `Encoding` literal is `myprotocol`). This procedure shall have one parameter of the same type (Charstring, Bitstring or Octetstring) as the corresponding procedure for encode, and shall return the choice type for the relevant path for the invocation (see below in *Semantics*).

Each encode or decode procedure shall be functional (that is, it shall not contain states and shall not change the value of any SDL variable external to the procedure when it is interpreted).

The length of the list of optional *Expression*s shall be the same as the number of *Sort-reference-identifier*s in the *Signal-definition* denoted by the *Signal-identifier*.

Each *Expression* in an *Encoding-expression* shall be sort compatible to the corresponding (by position) *Sort-identifier-reference* in the *Signal-definition* denoted by the *Signal-identifier*.

For a *Channel-identifier* of an *Encoding-path* of an agent's *Encoding-expression,* the channel of this path shall be reachable with the *Signal-identifier* from the agent, and the *Channel-path* in the direction from the agent must include the *Signal-identifier* in its set of *Signal-identifier*s.

For a *Gate-identifier* of an *Encoding-path* of an agent's *Encoding-expression,* the gate shall be a gate of the agent or reachable via a channel with the *Signal-identifier* from the agent, and the *Out-signal-identifier-**set*** of the gate shall include the *Signal-identifier*.

The *Expression* in a *Decoding-expression* shall be compatible with the sort generated by an *Encoding-expression* using the same *Encoding-path* in a context where the context of the *Decoding-expression* is reachable via the *Encoding-path*.

Concrete grammar

<encoding rules> ::=
encode <rules identifier>

<rules identifier> ::=
<literal>

<encoding expression> ::=
encode { <signal identifier>[(<actual parameters>)] | <expression> }
[<encoding path>]

<encoding path> ::=
as { <channel identifier> | <gate identifier> }

<decoding expression> ::=
decode <expression> [<encoding path>]

The set of encoding rules of the path identified by the <channel identifier> or <gate identifier> is used.

If an <encoding expression> contains an <expression> (rather than a signal), the actual signal is derived from the <expression> as described in the model below. The sort of the <expression> shall be the sort of the implied choice data type corresponding to the set of signals for the <encoding path>.

The <encoding path> shall only be omitted from <encoding expression> if there is exactly one path with encoding for output of the signal in the context of the <encoding expression> and, in this case, the encoding for that path is used.

The <encoding path> shall only be omitted from <decoding expression> if there is exactly one path with encoding for input in the context of the decoding expression and, in this case, the encoding for that path is used.

An <encoding path> used as a <sort> denotes the implicit data defined for the path as defined below.

Semantics

The *Encoding-rules* item determines the set of rules used to change the implementation-dependent encoding for internal data:

– either to a standardized implementation-independent encoding (for one of the data type `Encoding` literals `text` to `EXER`),

– or to a defined implementation or application encoding (for a literal added to a specialization of the data type `Encoding`).

Encode is invoked whenever a signal is output via a path with a set of encoding rules specified and the corresponding encode procedure is called. When a signal is input from such a path, decode of the data into the internal encoding is invoked by calling the corresponding decode procedure. This encode and decode therefore has no impact on the semantics of SDL, as defined in ITU-T Rec. Z.100, but requires specific encoding of signals for the specified paths and, therefore, enables different parts of the system to be implemented separately.

When an *Encoding-expression* or *Decoding-expression* is interpreted, the appropriate procedure is called.

Encode relative to the set of encoding rules for a path is invoked in an *Encoding-expression* to produce a Charstring, Bitstring or Octetstring depending on the context and the set of encoding rules used. The Charstring, Bitstring or Octetstring produced by encode is decoded by a *Decoding-expression* using the same set of encoding rules for the same path. The set of encoding rules of the encoding path identified by the *Channel-identifier* or *Gate-identifier* or *Interface-identifier* is used.

For an *Encoding-expression*, the data is encoded as if it were going to be output on that path. The result is a data type corresponding to the set of encoding rules for the path.

For a *Decoding-expression*, the data is decoded as if it had been received on the specified path. The result is an expression corresponding to the implicit data type for input from that channel in the decode context as defined below. If decoding fails, the InvalidReference exception is raised.

For a path that has encoding, a data type is implicitly defined that corresponds to the SDL:

```
value type Implicitname /* an implicit and unique name */
{ choice
         signal1 value
         { struct
            1 Sort11 optional;
            2 Sort12 optional;
            3 Sort13 optional;
            /* ... and so on for each parameter of signal1 */
          } ;
         signal2 value
         { struct
            1 Sort21 optional;
            2 Sort22 optional;
            3 Sort23 optional;
            /* ... and so on for each parameter of the signal2 */
          } ;
         signal3 NULL;  /* no parameters */
         /* ... and so on for each signal */
}
```

where

`signal1`, `signal2` etc. are the names of the signals carried by the path, and `Sort11`, `Sort12` etc. are data types corresponding to the parameters of `signal1`, and `Sort21`, `Sort22` etc. are data types corresponding to the parameters of `signal2`.

For a signal that has no parameters, the data type is the predefined `NULL` data type.

If the path is bidirectional and a signal is carried in both directions, there is only one choice in the implicit data type for this signal.

The implicit identifier of this data type is denoted by the **as** <channel identifier> or <gate identifier> for the path, so that a legal variable declaration is:

```
dcl message as user_input;
```

where `user_input` is the name of a channel or gate with encoding and a valid assignment is:

```
message := decode encoded_value as user_input;
```

The following enumerated data type for the standardized set of encoding rules shall be added to the package Predefined as defined in D.3/Z.100 in order to support the encoding of SDL data:

```
value type Encoding
     { literals text, BER, CER, DER, APER, UPER, CAPER, CUPER, BXER, CXER, EXER }
```

which is used to denote the required set of encoding rules as follows:

`text` for the set of text encoding rules defined in this Recommendation and produces a Charstring;

`BER` for the set of Basic Encoding Rules of ASN.1 and produces an Octetstring;

`CER` for the set of Canonical Encoding Rules of ASN.1 and produces an Octetstring;

`DER` for the set of Distinguished Encoding Rules of ASN.1 and produces an Octetstring;

`APER` for the basic Aligned variant of the Packed Encoding Rules of ASN.1 and produces an Octetstring;

`UPER` for the basic Unaligned variant of the Packed Encoding Rules of ASN.1 and produces a Bitstring;

`CAPER` for the Canonical Aligned variant of the Packed Encoding Rules of ASN.1 and produces an Octetstring;

`CUPER` for the Canonical Unaligned variant of the Packed Encoding Rules of ASN.1 and produces a Bitstring;

`BXER` for the Basic variant of the XML Encoding Rules of ASN.1 and produces a Charstring;

`CXER` for the Canonical variant of the XML Encoding Rules of ASN.1 and produces a Charstring;

`EXER` for the Extended variant of the XML Encoding Rules of ASN.1 and produces a Charstring.

The synonym PER is added to the **package** Predefined as an alternative for APER as follows:

```
synonym PER Encoding = APER;
```

The operator `last` of the data type `Encoding` is redefined to an unknown name, and therefore cannot be accessed, so that an application or implementation is able to extend the data type `Encoding` without any change where only the above rules are used. The data type `Encoding` can be specialized adding additional literals.

Model

If an <encoding expression> contains an <expression>, the choice present is determined from the expression and the signal with the same name as the choice is output with the values of the signal parameters given by the expression.

10.7.1 The set of text encoding rules

The set of text encoding rules is provided so that information can be conveyed on communication paths by means of text strings between elements in the system, and between the system and the environment. The encoding of characters is not defined. Though the text strings are, in general, readable by humans, that is not the purpose of the text encoding rules.

Semantics

If the set of encoding rules is specified as `text` the result of an encoding is a `Charstring`.

The actual string is determined as follows:

LEFT CURLY BRACKET and RIGHT CURLY BRACKET { } delimit the values of data types and show where the values start and stop, except where they occur within the encoding of a `Charstring` in which case they represent the actual <left curly bracket> or <right curly bracket> characters of ITU-T Rec. Z.100;

COMMA characters are used to delimit elements within a list (for example in a **struct** encoding);

Otherwise the actual string of characters is determined for each data type as defined below and illustrated as the 'Generated `CharString'` in the examples.

A complete signal is encoded as a list of values and is treated as a **choice** encoding of the implied data type for the path with encoding.

The details of text encoding are given in Annex A.

10.7.2 The sets of encoding rules standardized in the ITU-T X.69x series

The use of one of the names `BER`, `CER`, `DER`, `APER`, `UPER`, `CAPER`, `CUPER`, `BXER`, `CXER` or `EXER` (corresponding to an X.69x series set of encoding rules – see 10.7 *Semantics* above) shall only be specified if the signals carried by the path are defined as an ASN.1 CHOICE data type. The values are, therefore, encoded according to this data type treated as an ASN.1 ABSTRACT-SYNTAX.

11 Behaviour

11.1 Start

Start is as defined in ITU-T Rec. Z.100.

11.2 State

If during the consideration of the signals on the input port, the signal being considered does not correspond to any of the signals valid for any of the states of the agent (for example because the message received cannot be decoded to a valid signal), the InvalidReference exception is raised.

11.3 Input

If an <encoded input> is given for a path (a channel or gate), the messages that can be received from that path are received and stored in the variable given in their encoded form. These signals cannot be specified in any other input or saved for the same state. If the same signals can be received from another path, they are still assigned to the variable, which has to be sort compatible with the encoding for the path. Although the model given below describes the signals as first being decoded and then encoded again, it is expected that real implementations will optimize this to copying the encoded value to the variable given in the <encoded input>.

Concrete grammar

<input list> ::=
<stimulus> { , <stimulus> }*
| <asterisk input list>
| <encoded input>

<encoded input> ::=
encode <variable> [<encoding path>]

The <encoding path> shall only be omitted from <encoded input> if there is exactly one path with encoding leading to the context of the input that has a set of encoding rules that produces the sort (Charstring, Octetstring or Bitstring) of the <variable>.

Semantics

If the signal specified in the input is received via a channel that has a set of encoding rules specified, the signal is decoded according to that set of encoding rules.

Model

For each of the signals that can be received from the channel or gate specified in an <encoded input>, there is an implied input. This implied input is equivalent to assigning the values conveyed by signal to implicit local variables of appropriate types followed by an implied task. This implied task assigns the <variable> the value of an encoding expression for the signal and values received from the implicit variables using the set of encoding rules for the path of the <encoded input>. After the implied task, the transition following encoded input is interpreted.

11.4 Priority Input

Semantics

If the signal specified in the priority input is received via a channel that has a set of encoding rules specified, the signal is decoded according to that set of encoding rules.

11.5 Continuous signal

Continuous signal is as defined in ITU-T Rec. Z.100.

11.6 Enabling condition

Semantics

If the signal specified is received via a channel that has a set of encoding rules specified, the signal is decoded according to that set of encoding rules.

11.7 Save

Save is as defined in ITU-T Rec. Z.100.

11.8 Implicit transition

Implicit transition is as defined in ITU-T Rec. Z.100.

11.9 Spontaneous transition

Spontaneous transition is as defined in ITU-T Rec. Z.100.

11.10 Label

Label is as defined in ITU-T Rec. Z.100.

11.11 State machine and Composite state

State machine and Composite state are as defined in ITU-T Rec. Z.100.

11.12 Transition

Transition is as defined in ITU-T Rec. Z.100.

11.13 Action

11.13.1 Task

Task is as defined in ITU-T Rec. Z.100.

11.13.2 Create

Create is as defined in ITU-T Rec. Z.100.

11.13.3 Procedure call

Procedure call is as defined in ITU-T Rec. Z.100.

11.13.4 Output

If an <expression output> is given for a path (a channel or gate), the expression is a choice value used to derive the signal to output. The choice sort corresponds to the set of signals for the path.

If an <encoded output> is given for a path (a channel or gate), the expression given is used as the signal to output. In the model given below the expression is decoded to check the signal being sent.

Concrete grammar

<output body> ::=
<output body item> {, <output body item> }*
<communication constraints>

<output body item> ::=
<signal identifier> [<actual parameters>]
<expression output>
| <encoded output>

<expression output> ::=
<expression>

<encoded output> ::=
encode <expression>

When an <expression output> is used, there shall be exactly one <via path> in the <communication constraints> and the sort of the <expression> of the <expression output> shall be the sort of the implied choice data type corresponding to the set of signals for this path.

When an <encoded output> is used, there shall be exactly one <via path> in the <communication constraints> and this <via path> shall specify a gate or channel that has a set of encoding rules that corresponds to the sort (Charstring, Octetstring or Bitstring) of the <expression> of the <encoding output>.

Semantics

If a signal is output via a path that has a set of encoding rules specified, the signal is encoded according to that set of encoding rules.

Model

If the <output body item> is an <expression output>, the choice present is determined from the expression and the signal with the same name as the choice is output with the values of the signal parameters given by the expression.

If the <output body item> is an <encoded output>, the signal that is output is the signal obtained from decoding the contents of the expression according to the decoding rule for receiving signals sent on the specified path. The signal selected is the signal with the same name as the choice present in the decoding. If the signal is not valid for the path, or decoding fails for any reason (for example if the string of the expression does not validly correspond to one of the signals for the path), the OutOfRange exception is raised and no signal is sent, otherwise the value of the decoding is output as the signal. Because the expression is already encoded as a string for the path, the string value can be sent with no further conversion.

11.13.5 Decision

Decision is as defined in ITU-T Rec. Z.100.

11.14 Statement list

Statement list is as defined in ITU-T Rec. Z.100.

11.15 Timer

Timer is as defined in ITU-T Rec. Z.100.

11.16 Exception

Exception is as defined in ITU-T Rec. Z.100.

12 Data

12.1 Data definitions

Data definitions are as defined in ITU-T Rec. Z.100 except that <sort> is extended with <encoding path>.

Concrete grammar

<sort> ::=
<basic sort> [(<range condition>)]
| <anchored sort>
| <expanded sort>
| <reference sort>
| <pid sort>
| <inline data type definition>
| <inline syntype definition>
| <encoding path>

12.2 Passive use of data

The abstract syntax of expressions is extended to include encoding and decoding, otherwise the passive use of data is as defined in ITU-T Rec. Z.100.

12.2.1 Expressions

Encoding and decoding are added as expressions.

Abstract grammar
Active-expression = *Variable-access*
| *Conditional-expression*
| *Operation-application*
| *Equality-expression*
| *Imperative-expression*
| *Range-check-expression*
| *Value-returning-call-node*
| *State-expression*
| *Encoding-expression*
| *Decoding-expression*

Concrete grammar

<expression0> ::=
<operand>
| <create expression>
| <value returning procedure call>
| <encoding expression>
| <decoding expression>

12.3 Active use of data

The abstract syntax of expressions is extended to include encoding and decoding, otherwise the active use of data is as defined in ITU-T Rec. Z.100.

13 Generic system definition

Generic system definition is as defined in ITU-T Rec. Z.100.

Annex A

Specification of the set of text encoding rules

The data types are presented below with data types of Annex D/Z.100 in the order they occur, followed by other data types.

A.1 Boolean

`Boolean` values, False and True shall be encoded as LATIN CAPITAL LETTER T and LATIN CAPITAL LETTER F respectively.

Example

```
dcl Var_Boolean Boolean;
task Var_Boolean := true;
```

Generated `CharString`: T

A.2 Character

`Character` values shall be encoded as the actual Character with the exception of the `ESC` character, which is encoded as two ESCAPE characters. An undefined or missing Character value shall be encoded as an ESCAPE character followed by the NULL character.

NOTE – The Generated `CharString` may contain non-printing characters.

Example

```
dcl Var_Character Character;
task Var_Character := 'M';
```

Generated `CharString`: M

A.3 String

`String` has a parameter for the item sort and shall be encoded as a list of values of the item sort enclosed in a LEFT CURLY BRACKET and a RIGHT CURLY BRACKET separated by COMMA characters.

Example

```
value type IntString inherits String <Integer>;
dcl str IntString;

task str:= ''//mkstring(6)//mkstring(9)//mkstring(1948);
```

Generated `CharString`: {6,9,1948}

A.4 Charstring, IA5String, NumericString, PrintableString, VisibleString

Although `Charstring` is based on String, because it is a predefined data type and is commonly used, it is given a special encoding. `IA5String` and `Charstring` cover the same range of characters and have the same encoding. `NumericString`, `PrintableString` and `VisibleString` are subsets of `IA5String`.

These string types shall be encoded as a string of characters enclosed in APOSTROPHE (') characters without change except the <apostrophe> (') which shall be encoded as two APOSTROPHE (') characters.

NOTE 1 – Within a character string <comma>s, <left curly brackets> and <right curly bracket>s are treated as normal characters.

NOTE 2 – The Generated `CharString` may therefore contain non-printing characters including end of file or end of record characters.

Example

```
dcl Var_Charstring Charstring;

task Var_Charstring := 'Fred''s world;
```

Generated `CharString`: 'Fred's world'

A.5 Integer

`Integer` shall be encoded as a decimal integer notation without leading DIGIT ZERO characters where negative numbers are immediately preceded by a HYPHEN-MINUS without leading DIGIT ZERO characters. Zero shall never be treated as a negative number.

Example

```
dcl i Integer;

task i := 2-7;
```

Generated `CharString`: –5

A.6 Natural

`Natural` is a **syntype** of `Integer` and shall therefore have the same encoding as `Integer`.

A.7 Real

The value 0.0 shall be encoded as DIGIT ZERO followed by FULL STOP followed by DIGIT ZERO.

Any negative value shall be encoded as a HYPHEN-MINUS immediately followed by the encoding of the value negated (a positive Real value).

A positive Real value shall be encoded as a single digit in the range 1 to 9, followed by a FULL STOP, followed by at least one and up to 11 decimal fraction digits, followed by LATIN SMALL LETTER E 'e', followed by an exponent. The exponent is the Integer encoding of the exponent value: the power to the base 10 to apply to the first part of the number. The exponent may be negative.

Trailing fractional zero digits can be omitted. Some Real values cannot be precisely encoded and it is possible that unequal Real values that have very small difference have the same encoding. In any case, whether a Real value is precisely correct within an application will depend on how Real has been implemented. For example, if the Real value 2.0/7.0 is actually stored as the ratio of two integers (2 and 7), this is absolutely precise, whereas a more conventional encoding could be 0.2857142857, which is only correct to 10 decimal places.

Examples

```
dcl p, q, r1, r2 Real := 2000.0, 7.0, 0, 0;

task r1:= p/q;
task r2:= q/p;
```

Generated `CharString` for r1: 2.85714285714e2

Generated `CharString` for r2: 3.5e-3

A.8 Array

`Array` has two parameters: an index sort and a component sort. Either sort can in principle be any sort, but usually the index sort has a finite number of values.

Where the index sort has a finite number of ordered values an `Array` shall be encoded as a list of element encoding values, one for each element, separated by COMMA characters within a single pair of LEFT CURLY BRACKET and RIGHT CURLY BRACKET characters.

Example

```
value type ABC {literals A, B, C};
value type A1 inherits Array <ABC, Integer>;
dcl avalue, bvalue A1;

task avalue:= (. 3 .);
task bvalue[A]:= 3;
task bvalue[B]:= 5;
task bvalue[C]:= 7;
```

Generated `CharString` for avalue:{3,3,3}

Generated `CharString` for bvalue:{3,5,7}

It is possible the index is not ordered (that is, the "<" operator is not defined, or for example the index sort is a **structure** or **choice**). It is also possible the index sort does not have a finite number of values (that is, no possible number of values is infinite for example `CharString` or `Real`). For unordered or infinite index sort, the `Array` shall be encoded as the most frequent value followed by pairs of values for each element. If two or more values occur with the highest frequency one of these is chosen on an arbitrary basis as the most frequent value. Each pair shall be between a LEFT CURLY BRACKET and a RIGHT CURLY BRACKET and separated by a COMMA: the first value is an index value and the second value is element value. No index values in the pairs shall be repeated.

Example

```
value type Dehashing inherits Array <CharString, CharString> := (.''.);
/* note that the default value is an empty string */
dcl hashtable Dehashing;

task htable ('ac'):= 'action';
task htable ('ab'):= 'ability';
task htable ('zzzz'):= 'end of document';
```

Generated `CharString` for `htable`: {'',{'ab','ability'},{'ac','action'},{'zzzz','end of document'}}

A.9 Vector

Vector is a special case of Array that always has index sort that is a subset of Natural with a range from 1 to a specified maximum value and any item sort. Vector shall therefore be encoded in the same way as an Array with a finite number of ordered index values: that is a list of element encoding values, one for each element, separated by COMMA characters within a single LEFT CURLY BRACKET and RIGHT CURLY BRACKET pair.

A.10 Powerset

A `Powerset` of a sort represents a mathematical set whose elements are all members of the sort. Where the sort has a finite number of ordered values, the `Powerset` of the sort shall be encoded as a bit string (a string of DIGIT ZERO and DIGIT 1 characters) enclosed in APOSTROPHE (') characters. Each bit position in this bit string represents if a value of the sort is present or absent in the set. A DIGIT 1 represents that the value is present and a DIGIT ZERO that the value is absent. The leftmost bit represents the smallest value of the element sort, and each other bit represents a value of the element sort larger than values for the bits to the left.

Example

```
value type Shortalpha {literals a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v};
/* note Shortalpha has 22 values */
value type Psa inherits Powerset<Shortalpha>;
dcl letters_used Psa;

task letters_used := (. h, c, u, r .);
```

Generated CharString: '0010000100000000010010'

Where the sort does not have a finite number of ordered values, the Powerset of the sort shall be encoded as the encoding of each element value present separated by COMMA characters enclosed in a LEFT CURLY BRACKET and RIGHT CURLY BRACKET pair. The element values may be in any order.

Example

```
value type Pchrstr inherits Powerset<Charstring>;
dcl strings_used Pchrstr;

task strings_used := (. 'me', 'you', 'us', 'me', 'again', 'hey', 'you' .);
```

Generated CharString: {'again','hey','you','me','us'}

A.11 Duration

Duration is used to denote 'a time interval'. Unless specified otherwise, the default value of the unit of Duration is one second.

Duration values shall be encoded as a pair of integers separated by a COMMA enclosed in a LEFT CURLY BRACKET and RIGHT CURLY BRACKET pair: the first integer give the number of units and the second integer any fractional part in nano (10^{-9}) units. By default these are seconds and nano-seconds. The value may be negative, in which case the encoding of the negated value shall be used with a HYPHEN-MINUS inserted immediately before the first integer.

Example

```
dcl dvar Duration;

task dvar := -17.00000007;
```

Generated CharString: {-17,700}

A.12 Time

Time is used to denote 'a point in time'. The value of a unit of Time shall be the same as the value of the unit of Duration. The origin of Time units is not specified by this Recommendation, but corresponds to the system clock being at zero: that is, NOW gives the value 0.0. It is allowed for Time values to be negative.

Time values shall be encoded in the same way as Duration.

Example

```
dcl tvar Time;

task tvar:= 17.0000017;
```

Generated CharString: {17,17000}

A.13 Bag

Bag shall be encoded in the same way as a Powerset with an element sort that does not have a finite ordered set of values, but with each element value preceded by an integer followed by COLON. The integer is the number of times the element sort occurs in the Bag value.

Example

```
value type B1 inherits Bag <Integer>;

dcl Var_Bag B1;

task Var_Bag := (. 7, 4, 7 .);
```

Generated CharString: {2:7,1:4}

A.14 Bit, Bitstring

Bit shall be encoded as a single DIGIT ZERO or DIGIT 1 representing a zero and one bit respectively.

Example

```
dcl Var_Bit Bit;

task Var_Bit := 1;
```

Generated CharString: 1

Bitstring shall be encoded as sequence of bits represented by DIGIT ZERO and DIGIT 1 characters enclosed in APOSTROPHE (') characters.

Example

```
dcl Var_Bit Bit_String;

task Var_Bit := '01011'B;
```

Generated CharString: '01011'

A.15 Octet, Octetstring

Octet shall be encoded as two characters corresponding to the hexadecimal notation of the Octet. The alphabetic characters shall be represented by lowercase letters (that is LATIN SMALL LETER A to LATIN SMALL LETTER F).

Example

```
dcl oct Octet;

task oct := 62;
```

Generated CharString: 3e

Octetstring is a sequence of Octet values and shall be encoded as a string of hexadecimal character pairs enclosed in APOSTROPHE (') characters. The alphabetic characters shall be represented by lowercase letters.

Example

```
dcl os Octetstring;

task os:= '12B32D'H;
```

Generated CharString: '12b32d'

A.16 Pid, pid sorts

To allow flexibility between applications pid values shall be encoded as a CHOICE value corresponding to the choice definition:

```
{ choice
      0 ApplicationDefined;
      1 Integer;
      2 OctetString;
      3 BitString;
      4 CharString;
      5 { struct
           identity CharString;
           instance Natural;
        };
}
```

A value of a pid sort shall be encoded as the corresponding value of the Pid sort.

All pid values shall be encoded using the same choice field as defined above: that is, if one pid value in an SDL model is encoded using the choice 1 Integer, all other pid values in the same model shall be encoded using choice 1 Integer. Otherwise which of the above choices is selected is application dependent. The first choice allows an application defined encoding to be used. The `ApplicationDefined` sort is not defined by this Recommendation.

The same agent instance of a model shall always have the same pid value encoding. Two different agent instances shall always have different encoded pid values. Otherwise the derivation of the encoded pid values is not further defined by the Recommendation.

Example (using choice 5)

```
dcl agent_id Pid; /* in second instance of process IPS */

task agent_id := self;
```

Generated `CharString`: {5, {IPS, 2}}

A.17 Null

The value Null (see ITU-T Rec. Z.105) shall be encoded as a DIGIT ZERO character.

Example

```
dcl Var_Null Null;

task Var_Null := Null;
```

Generated `CharString`: 0

A.18 Enumerated (literal list)

An enumerated type defined by a literal list or as an ASN.1 ENUMERATED type shall be encoded as the Natural value given by the application of `num` operator of the data type to the literal representing the value to be encoded.

Example

```
value type Enum
{ literals e1, e2, e3;
};

dcl evar Enum;

task evar := e2;
```

Generated `CharString`: 1

A.19 Structures

A structure data type value shall be encoded as a list of values for the fields within a LEFT CURLY BRACKET and RIGHT CURLY BRACKET pair separated by COMMA characters.

Example

```
value type Record { struct
f1 Integer;
f2 Charstring;
f3 Integer;};
dcl locat Record;
task locat:= (. 17, 'mid-field', 230125 .);
```

Generated `CharString`: {1, 'mid-field', 230125}

A.20 Choice

A choice value shall be encoded as a pair of values separated by a COMMA enclosed in a LEFT CURLY BRACKET and RIGHT CURLY BRACKET pair. The first value is the selected field indicated by name of the choice. The second value is the `CharString` for the field value according to the type of the field.

Example

```
value type C {choice
cs CharString;
cb Boolean;
};

dcl ChoiceVar C;

task ChoiceVar.cb := true;
```

Generated `CharString`: {cb,T}

A.21 Inherits and syntype

A **syntype** defines a new name for a data type with an optional constraint on the set values. The text encoding is therefore identical to that of the data type.

A data type that inherits from another data type has the same encoding as the parent data type, though in the case that additional literal or fields are added, these are added to the encoding.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
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, Internet protocol aspects and Next Generation Networks
Series Z	**Languages and general software aspects for telecommunication systems**

Geneva, 2005