TECHNICAL REPORT

ISO/IEC TR 10024

First edition
1992-07-01

Information technology — Telecommunications and information exchange between systems — Formal description of ISO 8073 (Classes 0, 1, 2, 3) in LOTOS

Technologies de l'information — Télécommunications et échange d'informations entre systèmes — Description formelle de l'ISO 8073 (classes 0, 1, 2 et 3) en LOTOS

Reference number
ISO/IEC TR 10024:1992(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) together form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The main task of technical committees is to prepare International Standards, but in exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

- type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;
- type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;
- type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ('state of the art', for example).

Technical reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC TR 10024, which is a technical report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Annexes A, B and C form an integral part of this Technical Report. Annexes D, E and F are for information only.

Introduction

In view of the complexity and widespread use of Open Systems Interconnection standards it is imperative to have precise and unambiguous definitions of these standards. Formal Description Techniques form an effective approach to providing such definitions. The use of Formal Description Techniques in this area, however, is relatively new, and their application on a wide scale requires an appropriate habituation period. Formal descriptions should be introduced gradually in standards if initially the number of National Bodies that are able to contribute to their development is too small, thus allowing time to gain experience an to develop education material.

Within ISO/IEC JTC 1/SC 6 an ad-hoc group for the formal description in LOTOS of the Transport Standards, viz. the Transport Service ISO 8072 and the Transport Protocol ISO 8073, was established in October 1985. The Formal Description Technique LOTOS, defined in ISO 8807, was still under development at that time within ISO TC 97/SC 21. Following the position of a majority of National Bodies, SC6 decided in January 1988 to progress both formal descriptions as Type 2 Technical Reports. The main reason for not incorporating them into the standards was that National Bodies expressed their current lack of expertise on the subject of Formal Description Techniques. It seemed therefore appropriate to let a period of time pass, during which the formal descriptions can be read and compared with the standards, and after which the status of the formal descriptions can be re-evaluated.

The purpose of this Technical Report is to provide a complete, consistent and unambiguous description of classes 0,1,2 and 3 of ISO 8073. It forms therefore a companion document to ISO 8073. This Technical Report has successfully passed correctness and completeness tests with respect to syntax and static semantics, according to the requirements of clause 3 of ISO 8807. Tools are available that provide interactive simulation facilities, so that confidence can be gained about the correctness of the description with respect to the dynamic semantics (e.g. deadlock-freedom), and about its correctness and completeness with respect to representation of intended behaviour requirements. These tools are still at the stage of prototypes, however.

This Technical Report contains the description of classes 0,1,2,3. Class 4 and NCMS are not yet included. This is due to the limited amount of resources allocated in JTC 1/SC 6 to the work on formal descriptions.

Significant contributions for the application of LOTOS in this work have been provided by Universities and research laboratories.

This page intentionally left blank

Information technology – Telecommunications and information exchange between systems – Formal description of ISO 8073 (Classes 0,1,2,3) in LOTOS

1 Scope

This Technical Report describes the classes 0,1,2 and 3 of the OSI Transport protocol, defined in ISO 8073 using the formal description technique LOTOS, which is defined in ISO 8807. Transport protocol class 4 and NCMS are not covered.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this Technical Report. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of the currently valid International Standards.

ISO 7498:1984, *Information Processing Systems – Open Systems Interconnection – Basic Reference Model.*

ISO 8072:1986, *Information Processing Systems – Open Systems Interconnection – Transport service definition.*

ISO/IEC 8073[1)], *Information Technology – Telecommunications and information exchange between systems – Connection oriented transport protocol specification.*

ISO TR 8509:1987, *Information Processing Systems – Open Systems Interconnection – Service conventions*

ISO 8807:1989, *Information Processing Systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour.*

ISO/IEC TR 10023:1992, *Information Technology – Telecommunications and information exchange between systems – Formal description of ISO 8072 in LOTOS.*

3 Definitions

The definitions contained or referred to in clause 3 of ISO 8073 are adopted in this Technical Report.

4 Symbols and abbreviations

This Technical Report uses the symbols defined in clause 6 (formal syntax) and Annex A (data type library) of ISO 8807.

The symbols and abbreviations contained in clause 4 of ISO 8073 are employed in this Technical Report. When used without reference, 'the protocol' and 'the service' refer to ISO 8073 resp. ISO 8072.

Usage of other symbols and abbreviations is explained upon their first occurrence.

5 Conventions

Informal explanations precede the formal definitions to which they refer, separated from them by a line identical to the first line below. Separation of formal definitions from subsequent informal explanations is indicated by a line identical to the second line below.

---*)

(*---

NOTE – This convention is consistent with the rules on delimitation of comments defined for LOTOS in ISO 8807. The formal text is set in Courier and the LOTOS keywords and operators are in boldface. Identifiers from the formal text are set in italics when used in the non-formal text.

The conventions defined in ISO TR 8509 are adopted, but together with the following one: the term "Request" refers both to request and to response service primitives, and the term "Indication" refers both to indication and to confirm service primitives.

---

1) Under revision

The conventions defined in ISO TR 8509 are adopted, but together with the following one: the term 'Request' refers both to request and to response service primitives, and the term 'Indication' refers both to indication and to confirm service primitives.

Parentheses are omitted in this specification in the following cases, where their removal is allowed by ISO 8807:

a) In expressions where, due to the fact that binary infix operators associate to the left, no ambiguity arises from removal of parentheses (subclause 6.2.8 of ISO 8807). Thus, for example, the expression '*(x gt 0) and (y lt 5)*' is specified as '*x gt 0 and (y lt 5)*'.

b) In expressions where, due to the priority of LOTOS behavioural operators, no ambiguity arises from removal of parentheses. Thus, for example, '`g; (stop)`' is specified as '`g; stop`'.

The LOTOS standard allows the use of explicit sort specification for simple-expression using the 'ofsort' terms. Such 'ofsort' terms are present in the specification only in those cases where their absence would give rise to ambiguity.

6 Requirements

This Technical Report complies with the requirements defined by clause 3 of ISO 8807.

No conformity requirements are defined by this Technical Report, but the conformance requirements defined by ISO 8073 are represented formally in LOTOS.

7 Introduction of the formal description

The present specification is designed on basis of two major requirements, which together provide the appropriate framework for the formal representation of the transport protocol architecture by way of a formal specification of a generic transport protocol entity:

a) the formal specification is to be provably consistent with the formal description in LOTOS of the OSI connection-oriented transport service ISO 8072 (TS FD for short, that is available in ISO TR 10023), assuming a correct formal description in LOTOS of the OSI connection-oriented network service ISO 8348 (NS FD for short),

b) the specification applies to any transport protocol entity which is claimed to comply with the requirements defined in the conformance clause of the protocol.

Figure 1 shows the resulting structure of the specification.

A formal data type is provided for the implementor's declaration of classes and options that are defined in the conformance clause of the protocol. Such a declaration is formally represented by a parameter of the specification. A boolean-valued function can be applied to the value of this parameter, that determines whether or not the value satisfies the static conformance requirements of the protocol (see clause 9).

Only if the static conformance requirements are satisfied, does the specification describe active behaviour, viz. it provides the implementor or tester with an abstract model of the protocol dynamic conformance requirements (see clause 10)

specification	parameters:gates: t,n SAP addresses Implementation options
global type definitions:	library data types SAP address types Implementation option types Auxiliary data types
behaviour definition:	if static conformance requirements are satisfied by the implementation options then dynamic conformance requirements else no behaviour
where local definitions:	static conformance specification dynamic conformance specification
endspec	

Figure 1 - Structure of the specification

The *t* (resp. *n*) gate represents the transport (resp. network) service boundary accessed by the entity. It models the totality of TSAPs (resp. NSAPs) at which the entity interacts with the TS user (resp. NS provider). Each TSAP (resp. NSAP) is uniquely identified by a TSAP address (resp. NSAP address) out of the set *tas* (resp. *nas*).

Proper co-operation between session and transport entities (and transport and network entities, resp.), within one open system, is ensured by assigning the same tas to the session and transport entities (and the same nas to the transport and network entities, resp.) residing in the same open system. This implies that a transport entity cannot co-operate with a session entity if they reside in different open systems.

The event structure at gate *t* consists of a triple of values, resp. of sorts *TAddress, TCEI, TSP* (see clause 8). The first value identifies the TSAP where the interaction occurs. The second value identifies the TCEP, within that TSAP, where the interaction occurs. The third value is the Transport Service Primitive (TSP) executed in the interaction. A similar structure is shown by events at gate *n*.

These event structures are the same as in the respective service formal descriptions, with the exception that this Technical Report describes a more refined, non-atomic execution of T-DATA primitives. This is required by correctness and generality of representation of the requirements relating to segmenting and flow control procedures. Clearly, also the data type definition of TSPs needed corresponding refinements: the definition presented in this Technical Report is a conservative extension of the definition found in the formal description in LOTOS of the transport service.

Full account is taken of the multiplicity aspects of the protocol. The behaviour of a never terminating transport entity is described.

According to the definition of process *TPEntity* (see clause 10), some of the component processes describe constraints that apply to, and depend upon, the behaviour of the protocol entity at only one of the two service boundaries. This class of constraints will be referred to as 'service constraints' (see clause 11), whilst the term 'protocol constraints' will refer to those which are described by the other components (see clause 12).

The service constraints ensure, for instance, that the address component of an interaction at *t* (resp. *n*) is a member of the set *tas* (resp. *nas*), that the identification of a connection by means of a connection endpoint identifier is unique within the scope of any given address, that the entity is ready to accept and support at least one TC and at least one NC, and so on.

The further decomposition of the protocol constraints exploits the usage of internal gates. Notice that, according to the formal semantics of LOTOS, the presence of internal gates in the specification by no way constrains the internal structure of any implementation, inasmuch as the specification requirements apply to the behaviour observable at service access points only.

The introduction of these internal gates and the definition of the related event structures (see clause 14) enable a clear separation of concerns between:

a) independent constraints on the provision of TCs (see clause 15, clause 16, clause 18, clause 20 through clause 24, clause 33 through clause 36, clause 39, clause 40),

b) independent constraints on the usage of NCs (see clause 17, clause 32, clause 37, clause 38),

c) dependencies between provision of TCs and usage of NCs, such as those relating to assignment of TCs to NCs (see clause 25), error recovery procedures (see clause 26 through clause 30), and association of TPDUs to TCs (see clause 31).

A distinction is made between the definition of abstract TPDU structures(see clause 19), where abstraction is made from the TPDU information that relates to TC identification, encoding, and encoded TPDUs, which are octet strings where all of the TPDU information is fully specified (see clause 40).

Some auxiliary definitions are finally presented (see clause 41), which do not directly represent constraints found in the protocol, but are frequently employed in the preceding formal definitions. Processes that have a name beginning with *Run* are all defined in clause 41.

Except for the definitions of the specification parameter types, definitions shared with the TS FD, ISO TR 10023, and sharable with a NS FD are presented in Annex A and Annex B respectively. The data types referred to in the **library** construct are imported from the LOTOS library of data types.

```
------------------------------------------------------------------------------*)
specification TransportProtocolEntity[t, n]
  (tas : TAddresses, nas : NAddresses, tpeo : TPEOptions) : noexit
library Set, String, NaturalNumber, NatRepresentations, DecNatRepr, OctetString, Octet, Bit,
  BitNatRepr, Boolean, FBoolean, Element
endlib
(*------------------------------------------------------------------------------
```

8 Specification parameters data types

8.1 General

Two of the three specification parameters (see clause 7) have simple structures: finite sets of addresses and are presented in 8.2.

The options parameter contains the description of the optional procedures and their parameters, viz. supported classes, supported roles (*Initiator* or *responder*), maximum TPDUsize for each supported class and the optional procedures listed in table 9 of the protocol.

It has a slightly more complicated structure, which can be summarized as follows: a value of sort *TPEOptions is* a 4-tuple (see 8.3), where the components correspond to the requirements mentioned in subclause 14.6 of the protocol. The component structures are respectively described in 8.3.1 through 8.3.4.

8.2 Address sets

Although some these definitions below are shared with the TS FD or can be shared with the NS FD, these definitions are presented here, instead of in clause A.2, resp clause B.2, to allow description of the specification parameters.

```
------------------------------------------------------------------------------*)
type TransportAddress
is GeneralIdentifier renamedby
sortnames
  TAddress for Identifier
opnnames
  SomeTAddress for SomeIdentifier
  AnotherTAddress for AnotherIdentifier
endtype (* TransportAddress *)
type TransportAddresses
is Set actualizedby TransportAddress using
sortnames
  TAddress for Element
  Bool for FBool
  TAddresses for Set
endtype (* TransportAddresses *)
type NetworkAddress
is GeneralIdentifier renamedby
sortnames
  NAddress for Identifier
opnnames
  SomeNAddress for SomeIdentifier
  AnotherNAddress for AnotherIdentifier
endtype (* NetworkAddress *)
type NetworkAddresses
is Set actualizedby NetworkAddress using
sortnames
  NAddress for Element
  Bool for FBool
  NAddresses for Set
endtype (* NetworkAddresses *)
(*------------------------------------------------------------------------------
```


8.3 Options

A value of sort *TPEOptions is* a 4-tuple constructed by the function *TPEOptions*. The functions *Classes*, *Roles*, *OptionalProcedures* and *MaxTPDUSizeMap* are projection functions, viz. application of each of them to a quadruple yields the value of one distinct component. The definition of boolean equality is straightforward.

```
------------------------------------------------------------------------------*)
type TPEOptions
is Classes, TPEntityRoles, OptionalProcedures, MaxTPDUSizeMap, PrMaxTPDUSize
sorts
  TPEOptions
opns
  TPEOptions                : Classes, TPERoles, OptionalProcedures,
  MaxTPDUSizeMap, PrMaxTPDUSize                    -> TPEOptions
  Classes                   : TPEOptions           -> Classes
  Roles                     : TPEOptions           -> TPERoles
  OptionalProcedures        : TPEOptions           -> OptionalProcedures
  MaxTPDUSizeMap            : TPEOptions           -> MaxTPDUSizeMap
  PrMaxTPDUSize             : TPEOptions           -> PrMaxTPDUSize
  _eq_, _ne_                : TPEOptions, TPEOptions -> Bool
eqns
forall
```

```
  t, t1, t2 : TPEOptions, c, c1 : Classes, r, r1 : TPERoles, p, p1 : OptionalProcedures,
  s, s1 : MaxTPDUSizeMap, pm, pm1 : PrMaxTPDUSize
ofsort Classes
  Classes(TPEOptions(c, r, p, s, pm)) = c;
ofsort TPERoles
  Roles(TPEOptions(c, r, p, s, pm)) = r;
ofsort OptionalProcedures
  OptionalProcedures(TPEOptions(c, r, p, s, pm)) = p;
ofsort MaxTPDUSizeMap
  MaxTPDUSizeMap(TPEOptions(c, r, p, s, pm)) = s;
ofsort PrMaxTPDUSize
  PrMaxTPDUSize(TPEOptions(c, r, p, s, pm)) = pm;
ofsort Bool
  TPEOptions(c, r, p, s, pm) eq TPEOptions(c1, r1, p1, s1, pm1)
   = c eq c1 and (r eq r1) and (p eq p1) and (s eq s1) and (pm eq pm1);
  t1 ne t2 = not(t1 eq t2);
endtype (* TPEOptions *)
```

(* ----------

8.3.1 Protocol classes

The five protocol classes are represented by corresponding constants of sort *Class*. These are ordered by means of the auxiliary function *h*, that maps classes to natural numbers. *Classes* defines finite sets of protocol classes, enriched with the function *HigherClasses* which yields the set of classes higher than a given class. This function is made use of in the description of class negotiation (see 20.2.2).

---------- *)


```
type Class
is NaturalNumber
sorts
  Class
opns
  Class0, Class1, Class2, Class3, Class4 :                    -> Class
  _eq_, _ne_, _le_, _lt_, _ge_, _gt_ : Class, Class           -> Bool
  h                                  : Class                  -> Nat
eqns
forall
  x, y : Class
ofsort Nat
  h(Class0) = 0;
  h(Class1) = succ(h(Class0));
  h(Class2) = succ(h(Class1));
  h(Class3) = succ(h(Class2));
  h(Class4) = succ(h(Class3));
ofsort Bool
  x eq y = h(x) eq h(y);
  x ne y = h(x) ne h(y);
  x le y = h(x) le h(y);
  x lt y = h(x) lt h(y);
  x ge y = h(x) ge h(y);
  x gt y = h(x) gt h(y);
endtype (* Class *)

type BasicClasses
is Set actualizedby Class using
sortnames
  Class for Element
  Bool for FBool
  Classes for Set
endtype (* BasicClasses *)

type Classes
is BasicClasses
opns
  HigherClasses              : Class                          -> Classes
eqns
forall
  c, d : Class
ofsort Classes
```

```
  HigherClasses(Class4) = {};
  h(d) = succ(h(c)) => HigherClasses(c) = Insert(d, HigherClasses(d));
endtype (* Classes *)
```

(* ———

8.3.2 Entity roles

An entity can either initiate CR TPDUs (*Initiator*), respond to CR TPDUs (*Responder*) or both (see 8.4 for *TwoTuplet*).

——— *)

```
type TPEntityRole
is TwoTuplet renamedby
sortnames
  TPERole for Tuplet
opnnames
  Initiator for TheOne
  Responder for TheOther
endtype (* TPEntityRole *)

type TPEntityRoles
is Set actualizedby TPEntityRole using
sortnames
  TPERole for Element
  Bool for FBool
  TPERoles for Set
endtype (* TPEntityRoles *)
```

(* ———

8.3.3 Optional procedures

OptionalProcedure defines constants for those procedures that have some 'optional' entry in table 9 of the protocol. The auxiliary function *h* enables a concise specification of boolean equality. The boolean functions *AppliesIn* specify whether a given optional procedure applies in a given class, or in some class of a given set of classes.

NOTE – The two functions *AppliesIn* differ in the second argument.

——— *)

```
type OptionalProcedure
is Classes, DecNatRepr
sorts
  OptionalProcedure
opns
  TPDUWithoutChecksum, NoFlowControl, ExtendedFormats, UseOfNSReceiptConfirmation,
    UseOfNSExpedited, UseOfSAck, UseOfRAck :              -> OptionalProcedure
  h                          : OptionalProcedure          -> Nat
  _eq_, _ne_                 : OptionalProcedure, OptionalProcedure -> Bool
  _AppliesIn_                : OptionalProcedure, Class    -> Bool
  _AppliesIn_                : OptionalProcedure, Classes  -> Bool
eqns
forall
  p, p1 : OptionalProcedure, c : Class, cs : Classes
ofsort Nat
  h(TPDUWithoutChecksum) = 0;
  h(NoFlowControl) = succ(h(TPDUWithoutChecksum));
  h(ExtendedFormats) = succ(h(NoFlowControl));
  h(UseOfNSReceiptConfirmation) = succ(h(UseOfNSReceiptConfirmation));
  h(UseOfNSExpedited) = succ(h(UseOfNsReceiptConfirmation));
  h(UseOfSAck) = succ(h(UseOfNsExpedited));
  h(UseOfRAck) = succ(h(UseOfSAck));
ofsort Bool
  p eq p1 = h(p) eq h(p1);
  p ne p1 = not(p eq p1);
  TPDUWithoutChecksum AppliesIn c = c eq Class4;
  NoFlowControl AppliesIn c = c eq Class2;
  ExtendedFormats AppliesIn c = c ge Class2;
  UseOfNSReceiptConfirmation AppliesIn c = c eq Class1;
  UseOfNSExpedited AppliesIn c = c eq Class1;
  UseOfSAck AppliesIn c = c eq Class4;
  UseOfRAck AppliesIn c = c eq Class1 or (c ge Class3);
```

```
  p AppliesIn {} = false;
  c IsIn cs, p AppliesIn c => p AppliesIn cs = true;
  c NotIn cs, not(p AppliesIn c) => p AppliesIn Insert(c, cs) = p AppliesIn cs;
endtype (* OptionalProcedure *)

type OptionalProcedures
is Set actualizedby OptionalProcedure using
sortnames
  OptionalProcedure for Element
  Bool for FBool
  OptionalProcedures for Set
endtype (* OptionalProcedures *)
```

(* ---

8.3.4 Maximum TPDU sizes

MaxTPDUSize defines seven constants that represent the values listed in subclause 14.6 of the protocol, and the functions *TPDUSize*, which yields the corresponding natural number, and *MaxTPDUSizeFor*, which yields the maximum TPDU size defined by the protocol for each class (see subclause 13.3.4.b of the protocol). Equality and order of the constants are respectively specified by means of equality and order of their corresponding *TPDUSize* value.

MaxTPDUSizes defines sets of *MaxTPDUSize* values, and the boolean functions *MaxIn*, which tells whether a given *MaxTPDUSize* value is the maximum in a given set of such values, and *Maximal*, which tells whether a given set of *MaxTPDUSize* values contains all the values lower than or equal to the maximum value that it contains.

MaxTPDUSizeInClass defines pairs < *MaxTPDUSize* x *Class* > (see 8.4 for the definition of the generic type *Pair*). *BasicMaxTPDUSizeMap* defines sets of these pairs as values of sort *MaxTPDUSizeMap*.

In fact, a value of sort *MaxTPDUSizeMap* can be viewed as a binary relation between *MaxTPDUSize* and *Class*: in this view, the type *MaxTPDUSizeMap* endows the basic type with the functions *Classes*, which yields the range of the argument relation, and *MaxTPDUSizesOf* which, for given class and relation, yields the subset of the domain of the given relation characterized by the given class.

NOTE – The expression *1 + (2 + Dec(8))* below denotes a string of decimal digits, where '+' denotes digit concatenation with a digit string. The corresponding natural number value (128, in this case) is obtained by application of the function *NatNum*.

--- *)

```
type MaxTPDUSize
is DecNatRepr, Class
sorts
  MaxTPDUSize
opns
  Max1, Max2, Max3, Max4, Max5, Max6, Max7 :                -> MaxTPDUSize
  MaxTPDUSizeFor              : Class                       -> MaxTPDUSize
  TPDUSize                    : MaxTPDUSize                 -> Nat
  _eq_, _ne_, _le_, _lt_, _ge_, _gt_ : MaxTPDUSize, MaxTPDUSize -> Bool
eqns
forall
  c : Class, m, m1 : MaxTPDUSize
ofsort MaxTPDUSize
  MaxTPDUSizeFor(Class0) = Max5;
  c ne Class0 => MaxTPDUSizeFor(c) = Max7;
ofsort Nat
  TPDUSize(Max1) = NatNum(1 + (2 + Dec(8)));
  TPDUSize(Max2) = TPDUSize(Max1) + TPDUSize(Max1);
  TPDUSize(Max3) = TPDUSize(Max2) + TPDUSize(Max2);
  TPDUSize(Max4) = TPDUSize(Max3) + TPDUSize(Max3);
  TPDUSize(Max5) = TPDUSize(Max4) + TPDUSize(Max4);
  TPDUSize(Max6) = TPDUSize(Max5) + TPDUSize(Max5);
  TPDUSize(Max7) = TPDUSize(Max6) + TPDUSize(Max6);
ofsort Bool
  m eq m1 = TPDUSize(m) eq TPDUSize(m1);
  m ne m1 = TPDUSize(m) ne TPDUSize(m1);
  m le m1 = TPDUSize(m) le TPDUSize(m1);
  m lt m1 = TPDUSize(m) lt TPDUSize(m1);
  m ge m1 = TPDUSize(m) ge TPDUSize(m1);
  m gt m1 = TPDUSize(m) gt TPDUSize(m1);
endtype (* MaxTPDUSize *)

type BasicMaxTPDUSizes
is Set actualizedby MaxTPDUSize using
```

```
sortnames
  MaxTPDUSize for Element
  Bool for FBool
  MaxTPDUSizes for Set
endtype (* BasicMaxTPDUSizes *)
type MaxTPDUSizes
is BasicMaxTPDUSizes
opns
  _MaxIn_                    : MaxTPDUSize, MaxTPDUSizes   -> Bool
  Maximal                    : MaxTPDUSizes                -> Bool
eqns
forall
  m, m1 : MaxTPDUSize, ms : MaxTPDUSizes
ofsort Bool
  m NotIn ms => m MaxIn ms = false;
  m MaxIn Insert(m, {}) = true;
  m NotIn ms, m1 MaxIn ms => m MaxIn Insert(m, ms) = m gt m1;
  Maximal({}) = true;
  Maximal(Insert(m, {})) = m eq Max1;
  m NotIn ms, m1 MaxIn ms, Maximal(ms)
   => Maximal(Insert(m, ms)) = TPDUSize(m) eq (TPDUSize(m1) + TPDUSize(m1));
endtype (* MaxTPDUSizes *)
type MaxTPDUSizeClassPair
is Pair actualizedby MaxTPDUSize, Class using
sortnames
  MaxTPDUSize for Element
  Class for Element2
  Bool for FBool
  MaxTPDUSizeInClass for Pair
opnnames
  MaxTPDUSize for First
  Class for Second
  MaxTPDUSizeInClass for Pair
endtype (* MaxTPDUSizeClassPair *)
type BasicMaxTPDUSizeMap
is Set actualizedby MaxTPDUSizeClassPair using
sortnames
  MaxTPDUSizeInClass for Element
  Bool for FBool
  MaxTPDUSizeMap for Set
endtype (* BasicMaxTPDUSizeMap *)
type MaxTPDUSizeMap
is BasicMaxTPDUSizeMap, MaxTPDUSizes, Classes
opns
  Classes                    : MaxTPDUSizeMap              -> Classes
  _MaxTPDUSizesOf_           : Class, MaxTPDUSizeMap       -> MaxTPDUSizes
eqns
forall
  m : MaxTPDUSize, ms : MaxTPDUSizeMap, c, c1 : Class
ofsort Classes
  Classes({}) = {};
  Classes(Insert(MaxTPDUSizeInClass(m, c), ms)) = Insert(c, Classes(ms));
ofsort MaxTPDUSizes
  c NotIn Classes(ms) => c MaxTPDUSizesOf ms = {};
  c MaxTPDUSizesOf Insert(MaxTPDUSizeInClass(m, c), ms) = Insert(m, c MaxTPDUSizesOf ms);
  c ne c1 => c MaxTPDUSizesOf Insert(MaxTPDUSizeInClass(m, c1), ms) = c MaxTPDUSizesOf ms;
endtype (* MaxTPDUSizeMap *)
(*-------------------------------------------------------------------------------
```

The preferred maximum tpdusize parameter is expressed as a multiple of 128 octets.

```
-------------------------------------------------------------------------------*)
type PrMaxTPDUSize
is BasicPrMaxTPDUSize, NatRepresentations
opns
  NatNum                     : PrMaxTPDUSize               -> Nat
  _le_                       : PrMaxTPDUSize, prMaxTPDUSize-> Bool
```

```
eqns
forall
  pr, pr1 : PrMaxTPDUSize
ofsort Nat
  NatNum(NoPreference) = 0;
  NatNum(Next128(pr)) = NatNum(pr) + NatNum(1 + (2 + Dec(8)));
ofsort Bool
  pr le pr1 = NatNum(pr) le NatNum(pr1);
endtype (* PrMaxTPDUSize *)
type BasicPrMaxTPDUSize
is GeneralIdentifier renamedby
sortnames
  PrMaxTPDUSize for Identifier
opnnames
  NoPreference for SomeIdentifier
  Next128 for AnotherIdentifier
endtype (* BasicPrMaxTPDUSize *)
(*
```

8.4 Auxiliary definitions

This clause contains the definitions of *TwoTuplet, Pair* and *GeneralIdentifier* needed in constructing the definitions from clauses 8.2 and 8.3.

Element2 is needed in the construction of *Pair* and is a renaming of the library type *Element*.

Pair is a tuple < *Element* x *Element2* > with extended equality and operations for extracting its components.

TwoTuplet specifies a data type whose sort (*Tuplet*) can have 2 distinct values (*TheOne* and *TheOther*) together with equality.

GeneralIdentifier specifies an infinite number of identifiers together with equality.

NOTE – The auxiliary function *h* is introduced to simplify the specification of operation specifying equality. This technique will be used throughout this Technical Report.

*)


```
type Element2
is Element renamedby
sortnames
  Element2 for Element
endtype (* Element2 *)
type Pair
is Boolean, Element, Element2
sorts
  Pair
opns
  Pair                    : Element, Element2          -> Pair
  First                   : Pair                       -> Element
  Second                  : Pair                       -> Element2
  _eq_, _ne_              : Pair, Pair                 -> Bool
eqns
forall
  e1 : Element, e2 : Element2, p, p1 : Pair
ofsort Element
  First(Pair(e1, e2)) = e1;
ofsort Element2
  Second(Pair(e1, e2)) = e2;
ofsort Bool
  p = p1 => p eq p1 = true;
  First(p) ne First(p1) => p eq p1 = false;
  Second(p) ne Second(p1) => p eq p1 = false;
  p ne p1 = not(p eq p1);
endtype (* Pair *)
type TwoTuplet
is Boolean, NaturalNumber
sorts
  Tuplet
opns
  TheOne, TheOther        :                            -> Tuplet
  _eq_, _ne_              : Tuplet, Tuplet             -> Bool
```