JTC 1

TECHNICAL REPORT

ISO/IEC TR 9571

First edition
1989-09-15

Information technology — Open Systems Interconnection — LOTOS description of the session service

Traitement de l'information — Interconnexion de systèmes ouverts — Description en LOTOS du service de session

Reference number
ISO/IEC/TR 9571 : 1989 (E)

Contents

		page
	Foreword	iii
	Introduction	iv
1	Scope	1
2	Normative references	1
3	Definitions	2
4	Symbols and abbreviations	2
5	Conventions	2
6	Introduction to the formal description	3
7	Global constraints of the session service	4
8	Provision of a single session connection	5
9	Local constraints at a session connection endpoint	6
	9.1 Interface data types	6
	9.2 Processes for local constraints	24
10	End-to-end constraints for a session connection	35
	10.1 Association data types	36
	10.2 Processes for end-to-end constraints	41
11	Identification of session connections	44
12	Acceptance of session connections	46
13	Backpressure flowcontrol	46

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) together form a system for worldwide standardization as a whole. 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 a technical committee is to prepare International Standards but in exceptional circumstances, the publication of a technical report of one of the following types may be proposed:

- type 1, when the necessary support within the technical committee cannot be obtained for the publication of an International Standard, despite repeated efforts;

- type 2, when the subject is still under technical development requiring wider exposure;

- 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 9571, which is a technical report of type 2, was prepared by ISO/IEC JTC 1, *Information technology*.

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 important approach for providing such definitions. The use of Formal Description Techniques in this area is however relatively new and their application on a wide scale cannot be expected overnight. Formal descriptions should be introduced gradually in standards if initially the number of Member Bodies that are able to contribute to their development is too small, thus allowing time to gain experience and to develop educational material.

An ad-hoc group for the formal description of the Session Layer, i.e. of the Session Service ISO 8326 and the Session Protocol ISO 8327, was established in November 1985. This group applied the Formal Description Technique LOTOS, defined in ISO 8807, which at that time was still under development. In September 1986 two Working Documents were produced which contained the LOTOS draft specifications of ISO 8326 and ISO 8327 respectively. As a byproduct, the group also produced a number of Defect Reports on the standards, most of which have been accepted and incorporated in the standards.

A Ballot was then issued requesting Member Bodies to state their position concerning the progression of the formal descriptions. Based on this Ballot, SC21 decided in June 1987 to progress both formal descriptions as Type 2 Technical Reports. The main reason for not incorporating them into the standards was that Member Bodies expressed their current lack of expertise on the subject. It seemed therefore appropriate that a period of time passed, during which the formal decriptions can be read and compared with the standards, and after which the status and progression of the formal descriptions can be re-evaluated.

The purpose of this Technical Report is to provide a complete, consistent and unambiguous description of ISO 8326. It forms therefore a companion document to ISO 8326. It takes account of the Defect Reports incorporated in the standard (annex A of ISO 8326), however, it does not necessarily take account of subsequent amendments or addenda to the standard.

Information technology - Open Systems Interconnection - LOTOS description of the session service

1 Scope

This Technical Report contains a formal description of the OSI Basic Connection Oriented Session Service defined in ISO 8326. The formal definitions presented in this Technical Report are expressed in the formal description technique LOTOS, which is defined in ISO 8807.

These formal definitions are applicable for use in formal descriptions in LOTOS of the OSI Connection Oriented Session Protocol defined in ISO 8327, namely ISO/IEC TR 9572, and of the OSI Connection Oriented Presentation Protocol defined in ISO 8823.

The formal description is not limited to a single session connection, but also describes the service instances that result from multiple session connections either in parallel or in sequence. It therefore also formalizes aspects of multiplicity which are not presented in ISO 8326 directly, but by way of reference to the OSI Basic Reference Model, ISO 7498.

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 agreement based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO 7498: 1984, *Information processing systems - Open Systems Interconnection - Basic Reference Model.*

ISO 8326: 1987, *Information processing systems - Open Systems Interconnection - Basic connection oriented session service definition.*

ISO 8327: 1987, *Information processing systems - Open Systems Interconnection - Basic connection oriented session protocol specification.*

ISO/TR 8509: 1987, *Information processing systems - Open Systems Interconnection - Service conventions.*

ISO 8807: 1988, *Information processing systems - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour.*

ISO 8823: 1988, *Information processing systems - Open Systems Interconnection - Connection oriented presentation protocol specification.*

ISO/IEC/TR 9572: 1989, *Information processing systems - Open Systems Interconnection - LOTOS description of the session protocol.*

ISO/IEC/TR 10023: - [1], *Information processing systems - Open Systems Interconnection - LOTOS description of the transport service.*

3 Definitions

For the purpose of this Technical Report the definitions given in ISO 8326 apply.

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, and uses the abbreviations contained in clause 4 of ISO 8326.

The following additional abbreviations are employed in this Techical Report:

SC	session connection
SCEP	session connection endpoint
SCEI	session connection endpoint identifier
SSP	session service primitive

5 Conventions

Clauses 6 through 13 of this Technical Report constitute LOTOS text. All informal explanations in these clauses form LOTOS comments. They are thus separated from the LOTOS specifications (of data types and dynamic behaviour) according to the rules for comment delimitation. Moreover, informal explanations precede the formal definitions to which they refer and contain a final line of only "-" characters. Informal explanations following formal definitions contain a first line of only "-" characters.

Formal definitions, as well as formal symbols and identifiers referenced in informal explanations, are printed in italics.

The conventions defined in ISO/TR 8509 are adopted.

[1] To be published.

(* start of LOTOS text ----------

6 Introduction to the formal description

The formal description relates to the dynamic behaviour observable at the SS boundary. The whole service boundary is formally represented by a single gate *s*. Events at *s* consist of three values, of sort *SAddress*, *SCEI* and *SSP*, respectively. The first value identifies the SSAP where the event occurs. The second value identifies the SCEI within that SSAP where the event occurs. The third value represents the SSP executed in the event.

NOTE - An event is an atomic form of interaction. SSPs are thus represented as atomic interactions, which is consistent with ISO 8326. However, for reasons of consistency with the formal description of the session protocol, where flow control and segmenting require a finer granularity of those SSPs that can carry an unlimited amount of user data, this representation may have to be changed.

The possibility of multiple concurrent and consecutive SCs is represented by the formal description. Since this multiplicity is not restricted other than by nondeterministic decisions to not service certain requests and by the requirement of unambiguous use of SCEIs, the behaviour described is that of a non-terminating SS-provider.

A constraint-oriented style is adopted for the specification: different "types" of constraints in the service definition are specified in separate processes; these processes are appropriately composed, either synchronized or unsynchronized, in the specification to represent the total set of constraints. The decomposition in separate constraints is done at several levels.

The top level decomposition shows a structuring of the global constraints of the session service into the following separate constraints:

a) The dynamic behaviour of all potential SCs (described by process *SConnections*). This constraint is explicitly not concerned with the constraints below, i.e. it assumes no limits on the SS-provider capacity and does not care about administrative concerns for connection identification.

b) The correct allocation of SCEIs (described by process *SCIdentification*). It seems to be an explicit choice of ISO 7498 not to specify any requirement on local identification of connection endpoints, other than the obvious one that connection endpoint identifiers locally provide the means of distinguishing connections at the same service access point. This constraint applies precisely to this requirement.

c) The acceptance vs. refusal of new SCs by the SS-provider (described by process *SCAcceptance*). In spite of the "independence" of concurrent connections, finiteness of resources implies that interdependencies may exist. Such interdependencies concern the availability of resources to new connections in the same end-system, i.e. the ability of the service provider to accept an additional connection.

d) The backpressure flow control exerted by the SS-provider (described by process *SBackpressure*). This constraint relates to the fact that existing connections may be temporary blocked by the service provider, e.g. for reasons of resource shortage.

Constraint a) above is further decomposed into constraints related to the dynamic behaviour of a single SC (described by process *SConnection*). Each of these, in turn, is decomposed into "local" and "end-to-end" constraints (described by process *SCEP* and process *SCEPAssociation*,

respectively). Local constraints are constraints which are local to a SCEP, whereas end-to-end constraints concern the end-to-end relations which result from the exchange via the SS-provider.

Figure 1 gives an overview of the processes used for the formal definition of the Session Service and their relations (some of the lowest level processes are omitted). It also indicates the clauses where the definition of these processes can be found.

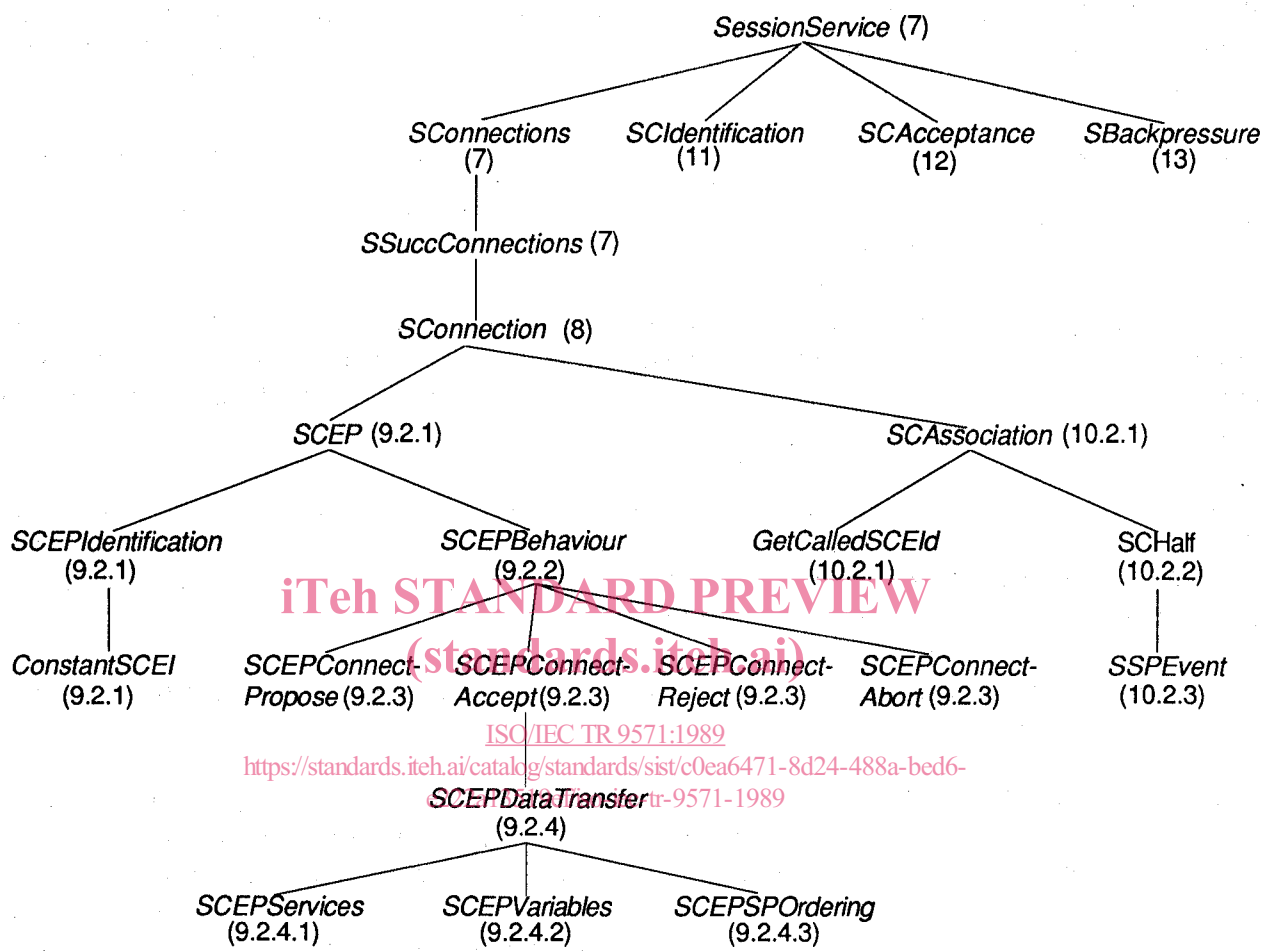

Figure 1 - Processes related by a tree structure: each "father" process contains (or is constructed from) one or more instances of its immediate "descendant" process(es)

The definition of data types precedes the definition of the dynamic behaviour in which these data types are used. Apart from standard data types, which are imported from the LOTOS library of data types, and some "ad-hoc" data types, two groupings of data types are identified, namely interface data types and association data types. Interface data types are used in the specification of local constraints, and are to be shared with the formal descriptions that may interwork with the present one, namely those of the Session Protocol and of the Presentation Protocol. Association data types are used in the specification of the end-to-end constraints; they abstract from the underlying protocol and are therefore probably not re-usable in other formal descriptions.

7 Global constraints of the session service

Standard data types for the construction of a boolean, a natural number, a generic set, a string of octets, and representations of a natural number (including the decimal representation) are imported from the LOTOS library of data types by means of the *library* construct.

The specification of the behaviour of the SS-provider consists of the conjunction of four separate constraints, as explained in clause 6. *SConnections* is described as being able to support a potentially infinite number of independent SCs. Concurrency is here represented by multiple instances of *SSuccConnections*, where each instance describes a succession of SCs. An SC is represented by a distinct instance of *SConnection*. Each instance of *SConnection* can terminate. The composites *SConnections* and *SSuccConnections* can never terminate because the possibility always exists that a new instance of *SConnection* is invoked. Thus, at any time, any number of SCs can be active. The SS-provider's nondeterminacy in limiting this potentially infinite concurrency is specified as a separate constraint, viz. by *SCAcceptance* (see clause 12). *SCIdentification* prescribes unique identification of SCEPs (see clause 11). *SBackpressure* enforces the constraint relating the SS-provider's backpressure (see clause 13).

```
----------------------------------------------------------------------------------------------------------------------*)
specification SessionService [s]: noexit

library FBoolean, Boolean, NaturalNumber, Element, Set, OctetString, NatRepresentations endlib

behaviour
SConnections [s] || SCIdentification [s] || SCAcceptance [s] || SBackpressure [s]
where

process SConnections [s]: noexit := SSuccConnections [s] ||| SConnections [s]
where
process SSuccConnections [s]: noexit := SConnection [s] >> SSuccConnections [s] endproc
endproc (* SConnections *)
(*----------------------------------------------------------------------------------------------------------------------
```


8 Provision of a session connection

Two separate constraints are distinguished with respect to the behaviour of a single SC, namely local constraints and end-to-end constraints. Local constraints apply to the possible behaviour at each SCEP taking only into account the history of SSPs executed at that SCEP. End-to-end constraints, on the other hand, relate the possible behaviour at each SCEP to the history of SSPs at the other SCEP (that is, the other end of the SC).

Local constraints are represented by two parallel instances of *SCEP* (see 9.2) that apply to the interaction with the Calling SS-user and with the Called SS-user, respectively. The type *SSUserRole* presents two constants that distinguish between these roles (see 9.1.5.1 for the definition of *Doublet*). End-to-end constraints are represented by an instance of *SCEPAssociation* (see 10.2).

The end of the SC lifetime is represented by termination of the two instances of *SCEP*; they force the instance of *SCEPAssociation*, a non-terminating process, to be disabled.

```
----------------------------------------------------------------------------------------------------------------------*)
type SSUserRole is Doublet renamedby
sortnames   SSUserRole for Doublet
opnnames    calling for constant1          called for constant2
endtype

process SConnection [s]: exit :=
( SCEP [s] (calling) ||| SCEP [s] (called) ) || ( SCAssociation [s] [> exit )
endproc
(*----------------------------------------------------------------------------------------------------------------------
```

9 Local constraints at a session connection endpoint

9.1 Interface data types

The interface data types consist of definitions for the construction of a Session (Service Access Point) address (see 9.1.1), a SCEI (see 9.1.2), and a SSP (see 9.1.3 and 9.1.4). A number of auxiliary definitions of general use are presented in 9.1.5.

Almost all type definitions include the definition of two infix boolean functions, *eq* and *ne*, for testing the equality and unequality, respectively, of two values of some sort.

9.1.1 Session address

No structure of the (Calling, Called or Responding) Session Address parameter is defined by ISO 8326. The following definition allows to represent an infinite number of Session Address values.

```
----------*)
type SessionAddress is Boolean
sorts SAddress
opns
someAddress: -> SAddress
AnotherAddress: SAddress -> SAddress
_eq_, _ne_: SAddress, SAddress -> Bool
eqns forall a,a1:SAddress ofsort Bool
a1 ne a = not (a1 eq a);                    someAddress eq someAddress = true;
someAddress eq AnotherAddress (a) = false;  AnotherAddress (a) eq someAddress = false;
AnotherAddress (a1) eq AnotherAddress (a) = a1 eq a;
endtype
(*----------
```


9.1.2 Session connection endpoint identifier

No structure of SCEIs is defined by ISO 8326. The following definition allows to represent an infinite number of SCEI values.

```
----------*)
type SCEndpointIdentifier is Boolean
sorts SCEI
opns
someCEI: -> SCEI
AnotherCEI: SCEI -> SCEI
_eq_, _ne_: SCEI, SCEI -> Bool
eqns forall e,e1:SCEI ofsort Bool
e1 ne e = not (e1 eq e);                    someCEI eq someCEI = true;
someCEI eq AnotherCEI (e) = false;          AnotherCEI (e) eq someCEI = false;
AnotherCEI (e1) eq AnotherCEI (e) = e1 eq e;
endtype
(*----------
```

9.1.3 Session service primitive

The specification of the SSP type is accomplished by way of a number of hierarchical type definitions. First, the basic construction of SSP values is presented (see 9.1.3.1), then a classification of SSPs (see 9.1.3.2), and subsequently a number of additional functions on SSPs (see 9.1.3.3 through 9.1.3.5). The individual SSP parameters are defined in 9.1.4.

9.1.3.1 Session service primitive basic construction

The construction of SSP values in type *BasicSSP* is a direct formulation of table 5 through 7 of ISO 8326. The functions that yield SSP values are referred to as "constructor" functions. This definition imports the definitions that relate to SSP parameters (see 9.1.4).

```
----------------------------------------------------------------------------------------------------------------------------*)
type BasicSSP is
SCIdentifier, SessionAddress, SQuality, SConnectResult, SRequirements, SSynchronizationNumber,
STokensAssignment, SData, STokens, SSyncMinorType, SResynchronizeType, SPExceptionReason,
SUExceptionReason, SActivityIdentifier, SReleaseResult, SPAbortReason
sorts SSP
opns
SCONreq, SCONind: SCRef, SAddress, SAddress, SQOS, SFUs, SSPSN, STsAss, SData -> SSP
SCONrsp, SCONcnf: SCRef, SAddress, SConResult, SQOS, SFUs, SSPSN, STsAss, SData -> SSP
SDTreq, SDTind, SEXreq, SEXind, STDreq, STDind, SCDreq, SCDind, SCDrsp, SCDcnf, SSYNMarsp,
      SSYNMacnf,SUABreq, SUABind, SACTErsp, SACTEcnf, SRELreq, SRELind: SData -> SSP
SGTreq, SGTind: STokens -> SSP
SPTreq, SPTind: STokens, SData -> SSP
SCGreq, SCGind, SACTIrsp, SACTIcnf, SACTDrsp, SACTDcnf: -> SSP
SSYNmreq, SSYNmind: SSyncType, SSPSN, SData -> SSP
SSYNmrsp, SSYNmcnf, SSYNMareq, SSYNMaind, SACTEreq, SACTEind: SSPSN, SData -> SSP
SRSYNreq, SRSYNind: SResynType, SSPSN, STsAss, SData -> SSP
SRSYNrsp, SRSYNcnf: SSPSN, STsAss, SData -> SSP
SPERind: SPExcReason -> SSP
SUERreq, SUERind: SUExcReason, SData -> SSP
SACTSreq, SACTSind: SActId, SData -> SSP
SACTRreq, SACTRind: SActId, SActId, SSPSN, SCRef, SData -> SSP
SACTIreq, SACTIind, SACTDreq, SACTDind: SUExcReason -> SSP
SRELrsp, SRELcnf: SRelResult, SData -> SSP
SPABind: SPAbReason -> SSP
endtype
(*----------------------------------------------------------------------------------------------------------------------------
```


9.1.3.2 Session service primitive classification

Type *SSPConstant* defines a classification of SSPs. Each class, or "type", of SSP is represented by a constant with a name similar to the name used in table 5 through 7 of ISO 8326 to represent the SSP. The auxiliary function *h* that maps these constants to natural numbers is defined in order to simplify the definition of equality on SSP classes (and on SSP values, see 9.1.3.4). The boolean function *IsConstantReq* determines whether its argument is a constant indicating a request or response SSP. Similarly, *IsConstantInd* characterizes indication or confirm SSPs. *Even* and *Odd* are auxiliary functions used in the definition of *IsConstantReq* and *IsConstantInd*. *Succ* is defined by the standard type *NaturalNumber* and yields the successor of a given natural number.

Type *SSPBasicClassifiers* is a functional enrichment of type *BasicSSP*. It defines a function *k* that yields the constant corresponding to its argument SSP (thus relating abbreviated SSP names to their full names; the abbreviated names correspond with those defined in annex A of ISO 8326, except *SSYNMa* which corresponds with SSYNM). Furthermore, "recognizer" functions are defined that test whether their argument SSP is of a certain class. There are functions to test whether a SSP is a request, indication, response, or confirm of a particular service. In addition, there are functions that test whether a SSP is either a request or indication, or either a response or confirm of a particular service. *IsReq* and *IsInd* recognize whether the argument SSP is respectively a request or response and an indication or confirm; *IsProvGenerated* recognizes whether the argument SSP is generated by the SS-provider.

```
----------*)
type SSPConstant is NaturalNumber
sorts SSPConstant
opns
S-CONNECTrequest, S-CONNECTindication, S-CONNECTresponse, S-CONNECTconfirm,
      S-DATArequest, S-DATAindication, S-EXPEDITED-DATArequest, S-EXPEDITED-DATAindication,
      S-TYPED-DATArequest, S-TYPED-DATAindication, S-CAPABILITY-DATArequest,
      S-CAPABILITY-DATAindication, S-CAPABILITY-DATAresponse, S-CAPABILITY-DATAconfirm,
      S-TOKEN-GIVErequest, S-TOKEN-GIVEindication, S-TOKEN-PLEASErequest,
      S-TOKEN-PLEASEindication, S-CONTROL-GIVErequest, S-CONTROL-GIVEindication,
      S-SYNC-MINORrequest, S-SYNC-MINORindication, S-SYNC-MINORresponse,
      S-SYNC-MINORconfirm, S-SYNC-MAJORrequest, S-SYNC-MAJORindication,
      S-SYNC-MAJORresponse, S-SYNC-MAJORconfirm, S-RESYNCHRONIZErequest,
      S-RESYNCHRONIZEindication, S-RESYNCHRONIZEresponse, S-RESYNCHRONIZEconfirm
      S-P-EXCEPTION-REPORTindication, S-U-EXCEPTION-REPORTrequest,
      S-U-EXCEPTION-REPORT-indication, S-ACTIVITY-STARTrequest, S-ACTIVITY-STARTindication,
      S-ACTIVITY-RESUMErequest, S-ACTIVITY-RESUMEindication, S-ACTIVITY-INTERRUPTrequest,
      S-ACTIVITY-INTERRUPTindication, S-ACTIVITY-INTERRUPTresponse,
      S-ACTIVITY-INTERRUPTconfirm, S-ACTIVITY-DISCARDrequest, S-ACTIVITY-DISCARDindication,
      S-ACTIVITY-DISCARDresponse, S-ACTIVITY-DISCARDconfirm, S-ACTIVITY-ENDrequest,
      S-ACTIVITY-ENDindication, S-ACTIVITY-ENDresponse, S-ACTIVITY-ENDconfirm,
      S-RELEASErequest, S-RELEASEindication, S-RELEASEresponse, S-RELEASEconfirm,
      S-U-ABORTrequest, S-U-ABORTindication, S-P-ABORTindication: -> SSPConstant
h: SSPConstant -> Nat                                   Even, Odd: Nat -> Bool
IsConstantReq, IsConstantInd: SSPConstant -> Bool
_eq_, _ne_: SSPConstant, SSPConstant -> Bool
eqns forall c,c1:SSPConstant, n:Nat
ofsort Nat
h(S-CONNECTrequest) = 0;
h(S-CONNECTindication) = Succ (h(S-CONNECTrequest));
h(S-CONNECTresponse) = Succ (h(S-CONNECTindication));
h(S-CONNECTconfirm) = Succ (h(S-CONNECTresponse));
h(S-DATArequest) = Succ (h(S-CONNECTconfirm));
h(S-DATAindication) = Succ (h(S-DATArequest));
h(S-EXPEDITED-DATArequest) = Succ (h(S-DATAindication));
h(S-EXPEDITED-DATAindication) = Succ (h(S-EXPEDITED-DATArequest));
h(S-TYPED-DATArequest) = Succ (h(S-EXPEDITED-DATAindication));
h(S-TYPED-DATAindication) = Succ (h(S-TYPED-DATArequest));
h(S-CAPABILITY-DATArequest) =  Succ (h(S-TYPED-DATAindication));
h(S-CAPABILITY-DATAindication) = Succ (h(S-CAPABILITY-DATArequest));
h(S-CAPABILITY-DATAresponse) = Succ (h(S-CAPABILITY-DATAindication));
h(S-CAPABILITY-DATAconfirm) = Succ (h(S-CAPABILITY-DATAresponse));
h(S-TOKEN-GIVErequest) = Succ (h(S-CAPABILITY-DATAconfirm));
h(S-TOKEN-GIVEindication) = Succ (h(S-TOKEN-GIVErequest));
h(S-TOKEN-PLEASErequest) = Succ (h(S-TOKEN-GIVEindication));
h(S-TOKEN-PLEASEindication) = Succ (h(S-TOKEN-PLEASErequest));
h(S-CONTROL-GIVErequest) = Succ (h(S-TOKEN-PLEASEindication));
h(S-CONTROL-GIVEindication) = Succ (h(S-CONTROL-GIVErequest));
h(S-SYNC-MINORrequest) = Succ (h(S-CONTROL-GIVEindication));
h(S-SYNC-MINORindication) = Succ (h(S-SYNC-MINORrequest));
h(S-SYNC-MINORresponse) = Succ (h(S-SYNC-MINORindication));
h(S-SYNC-MINORconfirm) = Succ (h(S-SYNC-MINORresponse));
h(S-SYNC-MAJORrequest) = Succ (h(S-SYNC-MINORconfirm));
h(S-SYNC-MAJORindication) = Succ (h(S-SYNC-MAJORrequest));
h(S-SYNC-MAJORresponse) = Succ (h(S-SYNC-MAJORindication));
h(S-SYNC-MAJORconfirm) = Succ (h(S-SYNC-MAJORresponse));
h(S-RESYNCHRONIZErequest) = Succ (h(S-SYNC-MAJORconfirm));
h(S-RESYNCHRONIZEindication) = Succ (h(S-RESYNCHRONIZErequest));
h(S-RESYNCHRONIZEresponse) = Succ (h(S-RESYNCHRONIZEindication));
h(S-RESYNCHRONIZEconfirm) = Succ (h(S-RESYNCHRONIZEresponse));
h(S-P-EXCEPTION-REPORTindication) = Succ (Succ (h(S-RESYNCHRONIZEconfirm)));
```

```
h(S-U-EXCEPTION-REPORTrequest) = Succ (h(S-P-EXCEPTION-REPORTindication));
h(S-U-EXCEPTION-REPORTindication) = Succ (h(S-U-EXCEPTION-REPORTrequest));
h(S-ACTIVITY-STARTrequest) = Succ (h(S-U-EXCEPTION-REPORTindication));
h(S-ACTIVITY-STARTindication) = Succ (h(S-ACTIVITY-STARTrequest));
h(S-ACTIVITY-RESUMErequest) = Succ (h(S-ACTIVITY-STARTindication));
h(S-ACTIVITY-RESUMEindication) = Succ (h(S-ACTIVITY-RESUMErequest));
h(S-ACTIVITY-INTERRUPTrequest) = Succ (h(S-ACTIVITY-RESUMEindication));
h(S-ACTIVITY-INTERRUPTindication) = Succ (h(S-ACTIVITY-INTERRUPTrequest));
h(S-ACTIVITY-INTERRUPTresponse) = Succ (h(S-ACTIVITY-INTERRUPTindication));
h(S-ACTIVITY-INTERRUPTconfirm) = Succ (h(S-ACTIVITY-INTERRUPTresponse));
h(S-ACTIVITY-DISCARDrequest) = Succ (h(S-ACTIVITY-INTERRUPTconfirm));
h(S-ACTIVITY-DISCARDindication) = Succ (h(S-ACTIVITY-DISCARDrequest));
h(S-ACTIVITY-DISCARDresponse) = Succ (h(S-ACTIVITY-DISCARDindication));
h(S-ACTIVITY-DISCARDconfirm) = Succ (h(S-ACTIVITY-DISCARDresponse));
h(S-ACTIVITY-ENDrequest) = Succ (h(S-ACTIVITY-DISCARDconfirm));
h(S-ACTIVITY-ENDindication) = Succ (h(S-ACTIVITY-ENDrequest));
h(S-ACTIVITY-ENDresponse) = Succ (h(S-ACTIVITY-ENDindication));
h(S-ACTIVITY-ENDconfirm) = Succ (h(S-ACTIVITY-ENDresponse));
h(S-RELEASErequest) = Succ (h(S-ACTIVITY-ENDconfirm));
h(S-RELEASEindication) = Succ (h(S-RELEASErequest));
h(S-RELEASEresponse) = Succ (h(S-RELEASEindication));
h(S-RELEASEconfirm) = Succ (h(S-RELEASEresponse));
h(S-U-ABORTrequest) = Succ (h(S-RELEASEconfirm));
h(S-U-ABORTindication) = Succ (h(S-U-ABORTrequest));
h(S-P-ABORTindication) = Succ (Succ (h(S-U-ABORTindication)));
ofsort Bool
Even (0) = true;                Even (Succ (0)) = false;        Even (Succ (Succ (n))) = Even (n);
Odd (n) = not (Even (n));       IsConstantReq (c) = Even (h(c)); IsConstantInd (c) = Odd (h(c));
c1 eq c = h(c1) eq h(c);        c1 ne c = not (c1 eq c);
endtype

type SSPBasicClassifiers is BasicSSP, SSPConstant
opns
k: SSP -> SSPConstant
IsSCONreq, IsSCONind, IsSCONrsp, IsSCONcnf, IsSCON, IsSCONAK, IsSDTreq, IsSDTind, IsSDT,
      IsSEXreq, IsSEXind, IsSEX, IsSTDreq, IsSTDind, IsSTD, IsSCDreq, IsSCDind, IsSCDrsp, IsSCDcnf,
      IsSCD, IsSCDAK, IsSGTreq, IsSGTind, IsSGT, IsSPTreq, IsSPTind, IsSPT, IsSCGreq, IsSCGind,
      IsSCG, IsSSYNmreq, IsSSYNmind, IsSSYNmrsp, IsSSYNmcnf, IsSSYNm, IsSSYNmAK,
      IsSSYNMareq, IsSSYNMaind, IsSSYNMarsp, IsSSYNMacnf, IsSSYNMa, IsSSYNMaAK,
      IsSRSYNreq, IsSRSYNind, IsSRSYNrsp, IsSRSYNcnf, IsSRSYN, IsSRSYNAK, IsSPERind,
      IsSUERreq, IsSUERind, IsSUER, IsSACTSreq, IsSACTSind, IsSACTS, IsSACTRreq, IsSACTRind,
      IsSACTR, IsSACTIreq, IsSACTIind, IsSACTIrsp, IsSACTIcnf, IsSACTI, IsSACTIAK, IsSACTDreq,
      IsSACTDind, IsSACTDrsp, IsSACTDcnf, IsSACTD, IsSACTDAK, IsSACTEreq, IsSACTEind,
      IsSACTErsp, IsSACTEcnf, IsSACTE, IsSACTEAK, IsSRELreq, IsSRELind, IsSRELrsp, IsSRELcnf,
      IsSREL, IsSRELAK, IsSUABreq, IsSUABind, IsSUAB, IsSPABind: SSP -> Bool
IsReq, IsInd, IsProvGenerated: SSP -> Bool
eqns forall r:SCRef, cg,cd,rg:SAddress, q:SQOS, cr:SConResult, rs:SFUs, sn:SSPSN, a:STsAss,
      d:SData, ts:STokens, st:SSyncType, rt:SResynType, per:SPExcReason, uer,er:SUExcReason,
      ai,aio:SActId, rr:SRelResult, ar:SPAbReason, p:SSP
ofsort SSPConstant
k(SCONreq(r,cg,cd,q,rs,sn,a,d)) = S-CONNECTrequest;
k(SCONind(r,cg,cd,q,rs,sn,a,d)) = S-CONNECTindication;
k(SCONrsp(r,rg,cr,q,rs,sn,a,d)) = S-CONNECTresponse;
k(SCONcnf(r,rg,cr,q,rs,sn,a,d)) = S-CONNECTconfirm;
k(SDTreq(d)) = S-DATArequest;                k(SDTind(d)) = S-DATAindication;
k(SEXreq(d)) = S-EXPEDITED-DATArequest;      k(SEXind(d)) = S-EXPEDITED-DATAindication;
k(STDreq(d)) = S-TYPED-DATArequest;          k(SDTind(d)) = S-TYPED-DATAindication;
k(SCDreq(d)) = S-CAPABILITY-DATArequest;     k(SCDind(d)) = S-CAPABILITY-DATAindication;
k(SCDrsp(d)) = S-CAPABILITY-DATAresponse;    k(SCDcnf(d)) = S-CAPABILITY-DATAconfirm;
k(SGTreq(ts)) = S-TOKEN-GIVErequest;         k(SGTind(ts)) = S-TOKEN-GIVEindication;
k(SPTreq(ts,d)) = S-TOKEN-PLEASErequest;     k(SPTind(ts,d)) = S-TOKEN-PLEASEindication;
```

k(SCGreq) = S-CONTROL-GIVErequest; *k(SCGind) = S-CONTROL-GIVEindication;*
k(SSYNmreq(st,sn,d)) = S-SYNC-MINORrequest; *k(SSYNmind(st,sn,d)) = S-SYNC-MINORindication;*
k(SSYNmrsp(sn,d)) = S-SYNC-MINORresponse; *k(SSYNmcnf(sn,d)) = S-SYNC-MINORconfirm;*
k(SSYNMareq(sn,d)) = S-SYNC-MAJORrequest; *k(SSYNMaind(sn,d)) = S-SYNC-MAJORindication;*
k SSYNMarsp(d)) = S-SYNC-MAJORresponse; *k(SSYNMacnf(d)) = S-SYNC-MAJORconfirm;*
k(SRSYNreq(rt,sn,a,d)) = S-RESYNCHRONIZErequest;
k(SRSYNind(rt,sn,a,d)) = S-RESYNCHRONIZEindication;
k(SRSYNrsp(sn,a,d)) = S-RESYNCHRONIZEresponse;
k(SRSYNcnf(sn,a,d)) = S-RESYNCHRONIZEconfirm;
k(SPERind(per)) = S-P-EXCEPTION-REPORTindication;
k(SUERreq(uer,d)) = S-U-EXCEPTION-REPORTrequest;
k(SUERind(uer,d)) = S-U-EXCEPTION-REPORTindication;
k(SACTSreq(ai,d)) = S-ACTIVITY-STARTrequest; *k(SACTSind(ai,d)) = S-ACTIVITY-STARTindication;*
k(SACTRreq(ai,aio,sn,r,d)) = S-ACTIVITY-RESUMErequest;
k(SACTRind(ai,aio,sn,r,d)) = S-ACTIVITY-RESUMEindication;
k(SACTIreq(er)) = S-ACTIVITY-INTERRUPTrequest;
k(SACTIind(er)) = S-ACTIVITY-INTERRUPTindication;
k(SACTIrsp) = S-ACTIVITY-INTERRUPTresponse; *k(SACTIcnf) = S-ACTIVITY-INTERRUPTconfirm;*
k(SACTDreq(er)) = S-ACTIVITY-DISCARDrequest; *k(SACTDind(er)) = S-ACTIVITY-DISCARDindication;*
k(SACTDrsp) = S-ACTIVITY-DISCARDresponse; *k(SACTDcnf) = S-ACTIVITY-DISCARDconfirm;*
k(SACTEreq(sn,d)) = S-ACTIVITY-ENDrequest; *k(SACTEind(sn,d)) = S-ACTIVITY-ENDindication;*
k(SACTErsp(d)) = S-ACTIVITY-ENDresponse; *k(SACTEcnf(d)) = S-ACTIVITY-ENDconfirm;*
k(SRELreq(d)) = S-RELEASErequest; *k(SRELind(d)) = S-RELEASEindication;*
k(SRELrsp(rr,d)) = S-RELEASEresponse; *k(SRELcnf(rr,d)) = S-RELEASEconfirm;*
k(SUABreq(d)) = S-U-ABORTrequest; *k(SUABind(d)) = S-U-ABORTindication;*
k(SPABind(ar)) = S-P-ABORTindication;
***ofsort** Bool*
IsSCONreq(p) = k(p) eq S-CONNECTrequest; *IsSCONind(p) = k(p) eq S-CONNECTindication;*
IsSCONrsp(p) = k(p) eq S-CONNECTresponse; *IsSCONcnf(p) = k(p) eq S-CONNECTconfirm;*
IsSDTreq(p) = k(p) eq S-DATArequest; *IsSDTind(p) = k(p) eq S-DATAindication;*
IsSEXreq(p) = k(p) eq S-EXPEDITED-DATArequest;
IsSEXind(p) = k(p) eq S-EXPEDITED-DATAindication;
IsSTDreq(p) = k(p) eq S-TYPED-DATArequest; *IsSTDind(p) = k(p) eq S-TYPED-DATAindication;*
IsSCDreq(p) = k(p) eq S-CAPABILITY-DATArequest;
IsSCDind(p) = k(p) eq S-CAPABILTY-DATAindication;
IsSCDrsp(p) = k(p) eq S-CAPABILITY-DATAresponse;
IsSCDcnf(p) = k(p) eq S-CAPABILITY-DATAconfirm;
IsSGTreq(p) = k(p) eq S-TOKEN-GIVErequest; *IsSGTind(p) = k(p) eq S-TOKEN-GIVEindication;*
IsSPTreq(p) = k(p) eq S-TOKEN-PLEASErequest; *IsSPTind(p) = k(p) eq S-TOKEN-PLEASEindication;*
IsSCGreq(p) = k(p) eq S-CONTROL-GIVErequest; *IsSCGind(p) = k(p) eq S-CONTROL-GIVEindication;*
IsSSYNmreq(p) = k(p) eq S-SYNC-MINORrequest; *IsSSYNmind(p) = k(p) eq S-SYNC-MINORindication;*
IsSSYNmrsp(p) = k(p) eq S-SYNC-MINORresponse; *IsSSYNmcnf(p) = k(p) eq S-SYNC-MINORconfirm;*
IsSSYNMareq(p) = k(p) eq S-SYNC-MAJORrequest;
IsSSYNMaind(p) = k(p) eq S-SYNC-MAJORindication;
IsSSYNMarsp(p) = k(p) eq S-SYNC-MAJORresponse;
IsSSYNMacnf(p) = k(p) eq S-SYNC-MAJORconfirm;
IsSRSYNreq(p) = k(p) eq S-RESYNCHRONIZErequest;
IsSRSYNind(p) = k(p) eq S-RESYNCHRONIZEindication;
IsSRSYNrsp(p) = k(p) eq S-RESYNCHRONIZEresponse;
IsSRSYNcnf(p) = k(p) eq S-RESYNCHRONIZEconfirm;
IsSPERind(p) = k(p) eq S-P-EXCEPTION-REPORTindication;
IsSUERreq(p) = k(p) eq S-U-EXCEPTION-REPORTrequest;
IsSUERind(p) = k(p) eq S-U-EXCEPTION-REPORTindication;
IsSACTSreq(p) = k(p) eq S-ACTIVITY-STARTrequest;
IsSACTSind(p) = k(p) eq S-ACTIVITY-STARTindication;
IsSACTRreq(p) = k(p) eq S-ACTIVITY-RESUMErequest;
IsSACTRind(p) = k(p) eq S-ACTIVITY-RESUMEindication;
IsSACTIreq(p) = k(p) eq S-ACTIVITY-INTERRUPTrequest;
IsSACTIind(p) = k(p) eq S-ACTIVITY-INTERRUPTindication;
IsSACTIrsp(p) = k(p) eq S-ACTIVITY-INTERRUPTresponse;
IsSACTIcnf(p) = k(p) eq S-ACTIVITY-INTERRUPTconfirm;

```
IsSACTDreq(p) = k(p) eq S-ACTIVITY-DISCARDrequest;
IsSACTDind(p) = k(p) eq S-ACTIVITY-DISCARDindication;
IsSACTDrsp(p) = k(p) eq S-ACTIVITY-DISCARDresponse;
IsSACTDcnf(p) = k(p) eq S-ACTIVITY-DISCARDconfirm;
IsSACTEreq(p) = k(p) eq S-ACTIVITY-ENDrequest;
IsSACTEind(p) = k(p) eq S-ACTIVITY-ENDindication;
IsSACTErsp(p) = k(p) eq S-ACTIVITY-ENDresponse;
IsSACTEcnf(p) = k(p) eq S-ACTIVITY-ENDconfirm;
IsSRELreq(p) = k(p) eq S-RELEASErequest;          IsSRELind(p) = k(p) eq S-RELEASEindication;
IsSRELrsp(p) = k(p) eq S-RELEASEresponse;         IsSRELcnf(p) = k(p) eq S-RELEASEconfirm;
IsSUABreq(p) = k(p) eq S-U-ABORTrequest;          IsSUABind(p) = k(p) eq S-U-ABORTindication;
IsSPABind(p) = k(p) eq S-P-ABORTindication;
IsSCON(p) = IsSCONreq(p) or IsSCONind(p);         IsSCONAK(p) = IsSCONrsp(p) or IsSCONcnf(p);
IsSDT(p) = IsSDTreq(p) or IsSDTind(p);            IsSEX(p) = IsSEXreq(p) or IsSEXind(p);
IsSTD(p) = IsSTDreq(p) or IsSTDind(p);            IsSCD(p) = IsSCDreq(p) or IsSCDind(p);
IsSCDAK(p) = IsSCDrsp(p) or IsSCDcnf(p);          IsSGT(p) = IsSGTreq(p) or IsSGTind(p);
IsSPT(p) = IsSPTreq(p) or IsSPTind(p);            IsSCG(p) = IsSCGreq(p) or IsSCGind(p);
IsSSYNm(p) = IsSSYNmreq(p) or IsSSYNmind(p);
IsSSYNmAK(p) = IsSSYNmrsp(p) or IsSSYNmcnf(p);
IsSSYNMa(p) = IsSSYNMareq(p) or IsSSYNMaind(p);
IsSSYNMaAK(p) = IsSSYNMarsp(p) or IsSSYNMacnf(p);
IsSRSYN(p) = IsSRSYNreq(p) or IsSRSYNind(p);      IsSRSYNAK(p) = IsSRSYNrsp(p) or IsSRSYNcnf(p);
IsSUER(p) = IsSUERreq(p) or IsSUERind(p);         IsSACTS(p) = IsSACTSreq(p) or IsSACTSind(p);
IsSACTR(p) = IsSACTRreq(p) or IsSACTRind(p);      IsSACTI(p) = IsSACTIreq(p) or IsSACTIind(p);
IsSACTIAK(p) = IsSACTIrsp(p) or IsSACTIcnf(p);    IsSACTD(p) = IsSACTDreq(p) or IsSACTDind(p);
IsSACTDAK(p) = IsSACTDrsp(p) or IsSACTDcnf(p);    IsSACTE(p) = IsSACTEreq(p) or IsSACTEind(p);
IsSACTEAK(p) = IsSACTErsp(p) or IsSACTEcnf(p);    IsSREL(p) = IsSRELreq(p) or IsSRELind(p);
IsSRELAK(p) = IsSRELrsp(p) or IsSRELcnf(p);       IsSUAB(p) = IsSUABreq(p) or IsSUABind(p);
IsReq(p) = IsConstantReq(k(p));                   IsInd(p) = IsConstantInd(k(p));
IsProvGenerated(p) = IsSPABind(p) or IsSPERind(p);
endtype
(*-------------------------------------------------------------------------------------------------------------------
```


9.1.3.3 Session service primitive parameter selectors

Type *SSPParameterSelectors* defines a further enrichment with functions that allow to determine the value of individual SSP parameters. Boolean functions are defined to test whether a given parameter value is carried in a given SSP. "Extractor" functions are defined to extract a parameter value from their argument SSP; extraction is only defined for those argument SSPs which are defined to carry the parameter to be extracted.

```
type SSPParameterSelectors is SSPBasicClassifiers
opns
_IsCRefOf_: SCRef, SSP -> Bool
_IsCallingOf_, _IsCalledOf_, _IsRespondingOf_: SAddress, SSP -> Bool
_IsQOSOf_: SQOS, SSP -> Bool                      _IsConResultOf_: SConResult, SSP -> Bool
_IsRqmsOf_: SFUs, SSP -> Bool                     _IsSPSNOf_: SSPSN, SSP -> Bool
_IsTsAssOf_: STsAss, SSP -> Bool                  _IsDataOf_: SData, SSP -> Bool
_IsTokensOf_: STokens, SSP -> Bool                _IsSyncTypeOf_: SSyncType, SSP -> Bool
_IsResynTypeOf_: SResynType, SSP -> Bool          _IsPExcReasonOf_: SPExcReason, SSP -> Bool
_IsUExcReasonOf_: SUExcReason, SSP -> Bool        _IsActIdOf_, _IsOldActIdOf_: SActId, SSP -> Bool
_IsRelResultOf_: SRelResult, SSP -> Bool          _IsPAbReasonOf_: SPAbReason, SSP -> Bool
CRef: SSP -> SCRef                                Called, Calling, Responding: SSP -> SAddress
QOS: SSP -> SQOS                                  ConResult: SSP -> SConResult
Rqms: SSP -> SFUs                                 SPSN: SSP -> SSPSN
TsAss : SSP -> STsAss                             Data: SSP -> SData
Tokens: SSP -> STokens                            SyncType: SSP -> SSyncType
ResynType: SSP -> SResynType                      PExcReason: SSP -> SPExcReason
UExcReason: SSP -> SUExcReason                    ActId, OldActId: SSP -> SActId
```