

Information technology — Language Independent Procedure Calling (LIPC)

ICS 35.060

National foreword

This British Standard reproduces verbatim ISO/IEC 13886 : 1996 and implements it as the UK national standard.

The UK participation in its preparation was entrusted to Technical Committee IST/5, Programming languages, their environments, and systems software interfaces, which has the responsibility to:

- aid enquirers to understand the text;
- present to the responsible international/European committee any enquiries on the interpretation, or proposals for change, and keep the UK interests informed;
- monitor related international and European developments and promulgate them in the UK.

A list of organizations represented on this committee can be obtained on request to its secretary.

From 1 January 1997, all IEC publications have the number 60000 added to the old number. For instance, IEC 27-1 has been renumbered as IEC 60027-1. For a period of time during the change over from one numbering system to the other, publications may contain identifiers from both systems.

Cross-references

The British Standards which implement international or European publications referred to in this document may be found in the BSI Standards Catalogue under the section entitled 'International Standards Correspondence Index', or by using the 'Find' facility of the BSI Standards Electronic Catalogue.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, the ISO/IEC title page, pages ii to vii, a blank page, pages 1 to 69 and a back cover.

Amendments issued since publication

Amd. No.	Date	Text affected

This British Standard, having been prepared under the direction of the DISC Board, was published under the authority of the Standards Board and comes into effect on 15 October 1997

© BSI 1997

ISBN 0 580 28571 5

INTERNATIONAL
STANDARD

ISO/IEC
13886

First edition
1996-03-15

**Information technology — Language-
Independent Procedure Calling (LIPC)**

*Technologies de l'information — Appel de procédure indépendant du
langage (LIPC)*



Contents

1	Scope	1
2	References	1
3	Definitions and Abbreviations	2
3.1	Definitions	2
3.2	Abbreviations	4
4	Conformance	5
4.1	Modes of conformance	5
4.1.1	Client mode conformance	5
4.1.2	Server mode conformance	5
5	A model of procedure calling: informal description	6
5.1	Model overview	6
5.2	Parameter passing	7
5.2.1	Methods of parameter passing	9
5.2.1.1	Call by Value Sent on Initiation	9
5.2.1.2	Call by Value Sent on Request	9
5.2.1.3	Call by Value Returned on Termination	10
5.2.1.4	Call by Value Returned when Available	10
5.2.2	Global data	10
5.2.3	Parameter Marshalling / Unmarshalling	11
5.2.4	Pointer Parameters	11
5.2.5	Private types	12
5.3	Execution-time Control	12
5.3.1	Terminations	12
5.3.1.1	Normal termination	12
5.3.1.2	Abnormal Termination	13
5.3.1.3	External Cancellation	13
5.3.1.4	Predefined conditions	13
5.4	Execution Control	14
5.4.1	Synchronous and Asynchronous Calling	14
5.4.2	Recursion	14
6	A model of procedure calling: formal description	14
6.1	Value	14

Descriptors: data processing, information interchange, computer software, programming languages, computer interfaces, models.

6.2	Boxes and global state	14
6.3	Symbol	15
6.4	Procedure image	16
6.5	Association	16
6.6	Procedure closures	17
6.7	Boxes, pointers, values, and datatypes	17
6.8	Interface closure	18
6.9	Interface type	19
6.10	Specifications	19
6.11	Basic procedure invocation	20
6.12	Type correctness	20
6.13	Associates	21
6.13.1	Simple Associates	21
6.13.2	Generalized Associates	22
6.14	Execution and Invocation contexts	23
6.15	Parameter translations	24
6.16	Defining Translation Procedures	26
7	Interface Definition Notation	27
7.1	Definitional Conventions	27
7.1.1	Character Set	27
7.1.2	Formal Syntax	27
7.1.3	Whitespace	28
7.2	Interface Type Declarations	29
7.2.1	Type references	29
7.2.2	Value References	30
7.3	Import Declarations	30
7.4	Value Declarations	31
7.5	Datatype Declarations	31
7.5.1	Primitive Datatypes	32
7.5.1.1	Integer	32
7.5.1.2	The real datatype	32
7.5.1.3	The character datatype	33
7.5.1.4	The boolean datatype	33
7.5.1.5	The enumerated datatype	33
7.5.1.6	The octet datatype	33
7.5.1.7	The procedure datatype	34
7.5.1.8	The state datatype	36
7.5.1.9	The ordinal datatype	36
7.5.1.10	The time datatype	37
7.5.1.11	The bit datatype	37
7.5.1.12	The rational datatype	37
7.5.1.13	The scaled datatype	37
7.5.1.14	The complex datatype	38
7.5.1.15	The void datatype	38
7.5.2	Generated datatypes	38
7.5.2.1	The record datatype	38
7.5.2.2	The choice datatype	39

7.5.2.3	The array datatype	39
7.5.2.4	The pointer datatype	40
7.5.3	Subtypes	40
7.6	Parameterized Types	40
7.7	Identifiers	41
7.7.1	Value references to fields	42
7.7.2	Value references to parameters, return-args, or to fields contained within them	42
7.7.3	Value references to formal-value-parms	43
7.7.4	Value references to value-expressions	43
7.7.5	Value references to enumeration-identifiers	43
7.7.6	Termination references	44

Annexes

A	Procedure Parameters	45
A.1	LIPC Reference / Local Access	45
A.2	LIPC Reference / LIPC Access	45
A.3	Local Reference / Local Access	46
B	Interface Definition Notation Syntax	47
C	How to do an LIPC binding for a language	53
C.1	Linking the client and the server	53
C.2	Client mode binding	54
C.3	Server mode binding	55
C.4	Procedure parameters	56
C.5	Global variables	57
D	LIPC IDN - RPC IDL Alignment overview	58
D.1	Interface Declarations	58
D.1.1	Attributes	58
D.1.2	Imports Clause	58
D.2	Other Declarations	59
D.2.1	Type Declarations	59
D.2.2	Value Declarations	59
D.2.3	Procedure Declarations	59
D.2.4	Termination Declarations	60
D.3	Primitive Datatypes	60
D.3.1	Boolean	60
D.3.2	State	60
D.3.3	Enumerated	60
D.3.4	Character	61
D.3.5	Ordinal	61
D.3.6	Time	61
D.3.7	Integer	61
D.3.8	Rational	62
D.3.9	Scaled	62
D.3.10	Real	62

D.3.11	Complex	63
D.3.12	Void	63
D.4	Type Qualifiers	63
D.5	Generated Datatypes	63
D.5.1	Choice	63
D.5.2	Pointer	64
D.5.3	Procedure	64
D.6	Aggregate Datatypes	64
D.6.1	Record	64
D.6.2	Set	64
D.6.3	Bag	65
D.6.4	Sequence	65
D.6.5	Array	65
D.6.6	Table	66
D.7	Derived Datatypes and Generators	66
D.7.1	Naturalnumber	66
D.7.2	Modulo	66
D.7.3	Bit	66
D.7.4	Bitstring	66
D.7.5	Characterstring	67
D.7.6	Timeinterval	67
D.7.7	Octet	67
D.7.8	Octetstring	67
D.7.9	Private	68
D.8	Other RPC Datatypes	68
D.9	"Dependent Values"	68
D.10	Cross References	68

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) 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 organizations 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. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 13886 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Annexes A to D of this International Standard are for information only.

Introduction

The purpose of this International Standard is to provide a common model for language standards for the concept of procedure calling. It is an enabling standard to aid in the development of language-independent tools and services, common procedure libraries and mixed language programming. In mixed language applications, server procedures would execute on language processors operating in server mode, and the procedures would be called from language processors operating in client mode. Note that the languages need not be different, and if the processors are the same the model collapses into conventional single processor programming.

Most programming languages include the concepts of procedures and their invocation. The main variance between the methods used in various programming languages lies in the ways parameters are passed between the client and server procedures. Procedure calling is a simple concept at the functional level, but the interaction of procedure calling with datatyping and program structure along with the many variations on procedure calling and restrictions on calling that are applied by various programming languages transforms the seemingly simple concept of procedure calling into a more complex feature of programming languages.

The need for a standard model for procedure calling is evident from the multitude of variants of procedure calling in the standardized languages. The existence of this International Standard for Language-Independent Procedure Calling (LIPC) does not require that all programming languages should adopt this model as their sole means of procedure calling. The nominal requirement is for programming languages to provide a mapping to LIPC from their native procedure calling mechanism, and to be able to accept calls from other programming languages who have defined a mapping to this International Standard.

This International Standard is a specification of a common model for procedure calling. It is not intended to be a specification of how an implementation of the LIPC is to be provided. Also, it is important to note that it does not address the question of how the procedure call initiated by the client mode processor is communicated to the server mode processor, or how the results are returned. The model defined in this International Standard is intended for use by languages so that they may provide standard mappings from their native procedure model. This International Standard depends on the International Standard for Language-Independent Datatypes, ISO/IEC 11404, for the definition of the datatypes that are to be supported in the model for LIPC that it provides.

Information technology – Language-Independent Procedure Calling (LIPC)

1 Scope

This International Standard specifies a model for procedure calls, and a reference syntax for mapping to and from the model. This syntax is referred to as the Interface Definition Notation. The model defined in this International Standard includes such features as procedure invocation, parameter passing, completion status, and environmental issues relating to non-local references and state.

This International Standard does not specify:

- the method by which the procedure call initiated by the client mode processor is communicated to the server mode language processor;
- the minimum requirements of a data processing system that is capable of supporting an implementation of a language processor to support LIPC;
- the mechanism by which programs written to support LIPC are transformed for use by a data processing system;
- the representation of a parameter.

NOTE – Originally it was the intention to align the definitions and concepts of this International Standard with those of the RPC standard (ISO/IEC 11578). Unfortunately, in a late stage of the development process of the RPC standard it was decided to use for that standard a completely different approach. Hence the intended alignment did not materialize.

Annex D gives an overview of the differences between the concepts as defined by this International Standard and the RPC standard.

2 Normative References

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

ISO 2375:1985, *Data processing – Procedure for registration of escape sequences.*

ISO/IEC 10646-1:1993, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane.*

ISO/IEC 11404:1996, *Information technology – Programming languages, their environments and system software interfaces – Language-independent datatypes.*

ISO/IEC 8824-1:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

ISO/IEC 8825-1:1995, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*

3 Definitions and Abbreviations

3.1 Definitions

For the purposes of this International Standard, the following definitions apply.

3.1.1 actual parameter: A value that is bound to a formal parameter during the execution of a procedure.

3.1.2 association: Any mapping from a set of symbols to values.

3.1.3 box: A model of a variable or container that holds a value of a particular type.

3.1.4 client interface binding: The possession by the client procedure of an interface reference.

3.1.5 client procedure: A sequence of instructions which invokes another procedure.

3.1.6 complete procedure closure: A procedure closure, all of whose global symbols are mapped.

3.1.7 configuration: Host and target computers, any operating system(s) and software used to operate a processor.

3.1.8 execution sequence: A succession of global states $s_1, s_2, \dots$ where each state beyond the first is derived from the preceding one by a single create operation or a single write operation.

3.1.9 formal parameter: The name symbol of a parameter used in the definition of a procedure to which a value will be bound during execution.

3.1.10 global state: The set of all existing boxes and their currently assigned values.

3.1.11 global symbol: Symbol used to refer to values that are permanently associated with a procedure.

3.1.12 implementation defined: An implementation defined feature is a feature that is left implementation dependent by this International Standard, but any implementation claiming conformity to this International Standard shall explicitly specify how this feature is provided.

3.1.13 implementation dependent: An implementation dependent feature is a feature which shall be provided by an implementation claiming conformity to this International Standard, but the implementation need not to specify how the feature is provided.

3.1.14 input parameter: A formal parameter with an attribute indicating that the corresponding actual parameter is to be made available to the server procedure on entry from the client procedure.

3.1.15 input/output parameter: A formal parameter with an attribute indicating that the corresponding actual parameters are made available to the server procedure on entry from the client procedure and to the client procedure on return from the server procedure.

3.1.16 interface closure: A collection of names and a collection of procedure closures, with a mapping between them.

3.1.17 interface execution context: The union of the procedure execution contexts for a given interface closure.

3.1.18 interface reference: An identifier that denotes a particular interface instance.

3.1.19 interface type: A collection of names and a collection of procedure types, with a mapping between them.

3.1.20 interface type identifier: An identifier that denotes an interface type.

3.1.21 invocation association: The invocation association of a procedure closure <Image, Association> applied to a set of actual parameter values is the association of the closure augmented by a mapping of all local symbols to values and all formal parameter symbols to the corresponding actual parameter values. Thus it is a binding to values of all symbols in the procedure image for the duration of the invocation.

3.1.22 invocation context: For a particular procedure call, the instance of the objects referenced by the procedure, where the lifetime of the objects is bounded by the lifetime of the call.

3.1.23 marshalling: A process of collecting actual parameters, possibly converting them, and assembling them for transfer.

3.1.24 output parameter: A formal parameter with an attribute indicating that the corresponding actual parameter is to be made available to the client procedure on return from the server procedure.

3.1.25 parameter: A parameter is used to communicate a value from a client to a server procedure. The value supplied by the client is the actual parameter, the formal parameter is used to identify the received value in the server procedure.

3.1.26 partial procedure closure: A procedure closure, some of whose global symbols are not mapped. Procedure closures may be complete, with all global symbols mapped, or partial with one or more global symbols not mapped.

3.1.27 procedure: The procedure value.

3.1.28 procedure call: The act of invoking a procedure.

3.1.29 procedure closure: A pair <procedure image, association> where the association defines the mapping for the image's global symbols and no others.

NOTE – Procedure closures are the values of procedure type referred to in ISO/IEC 11404 - Language-Independent Datatypes.

3.1.30 procedure execution context: For a particular procedure, an instance of the objects satisfying the external references necessary to allow the procedure to operate, where these objects have a lifetime longer than a single call of that procedure.

3.1.31 procedure image: A representation of a value of a particular procedure type, which embodies a particular sequence of instructions to be performed when the procedure is called.

3.1.32 procedure invocation: The object which represents the triple: procedure image, execution context, and invocation context.

3.1.33 procedure name: The name of a procedure within an interface type definition.

3.1.34 procedure return: The act of return from the server procedure with a specific termination.

3.1.35 procedure type: The family of datatypes each of whose members is a collection of operations on values of other datatypes. Note, this is a different definition from procedure value.

3.1.36 procedure value: A closed sequence of instructions that is entered from, and returns control to, an external source.

3.1.37 processor: A compiler or interpreter working in combination with a configuration.

3.1.38 server procedure: The procedure which is invoked by a procedure call.

3.1.39 symbol: A program entity used to refer to a value.

3.1.40 termination: A predefined status related to the completion of a procedure call.

3.1.41 unmarshalling: The process of disassembling the transferred parameters, possibly converting them, for use by the server procedure on invocation or by the client procedure upon procedure return.

3.1.42 value: The set Value contains all the values that might arise in a program execution.

3.2 Abbreviations

3.2.1 ASN.1: Abstract Syntax Notation - One

3.2.2 IDN: Interface Definition Notation

3.2.3 LID: Language-Independent Datatypes, as defined in ISO/IEC 11404:1995.

3.2.4 LIPC: Language-Independent Procedure Calling

4 Conformance

A language processor may conform to this International Standard by mapping its native procedure calling mechanism to the LIPC model that this International Standard defines.

NOTE – The term “language processor” used in this clause may be extended to include anything which processes information and contains a procedure calling mechanism.

4.1 Modes of conformance

A language processor claiming conformance to this International Standard shall conform in either or both of the following ways.

4.1.1 Client mode conformance

In order to conform in client mode, a language processor shall allow programs written in its language to call procedures written in another language and supported by another processor, using the language-independent procedure calling (LIPC) as provided by clauses 5, 6 and 7 of this International Standard. In this case it is said to conform in (and be able of operating in) client mode. As part of this, the language processor shall define a mapping from its own procedure calling model to the LIPC model.

NOTE – If a program using the LIPC facility is to be portable between processors which conform in client mode, the program and processors will also need to conform to the relevant language standard and the relevant standards binding for that language to the LIPC and LID standards.

4.1.2 Server mode conformance

In order to conform in server mode, a language processor shall allow programs written in another language to call procedures written in its language (i.e. it will accept and execute procedure calls generated by another processor which is executing in a program that is written in that other language and which is operating in client mode, and return control to that client processor upon completion), using the language-independent procedure calling (LIPC) as provided by clauses 5, 6 and 7 of this International Standard. In this case it is said to conform in (and be able of operating in) server mode. As part of this, the language processor shall define a mapping from the LIPC model to its own procedure calling model.

NOTES

- 1 It is also possible in principle for a client processor to use the model for procedure calls defined in this International Standard to call procedures in the same language; executing on a server processor in the same language, and if the processor conforms in both client and server mode, it is even possible for it to serve itself using this model.
- 2 If a procedure is to be portable between processors which conform in server mode and the procedure is still to be called by client processors and programs, the procedure, and the processors, will also need to conform to the relevant language standard and the relevant standards binding for that language to the LIPC and LID standards.

5 A model of procedure calling: informal description

5.1 Model overview

A procedure is defined to be a closed sequence of instructions that is entered from, and returns control to, an external source.

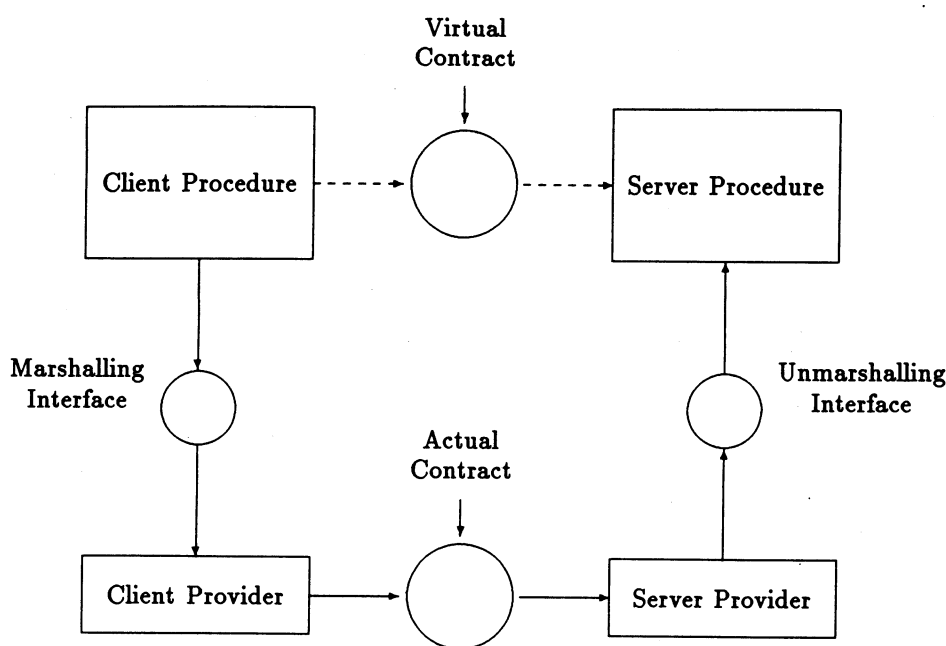
The general structure of a procedure call can be described as a single thread of execution in a particular program where the flow of control is passed from one procedure to another. The originator of the call is known as the client procedure and the procedure being called is referred to as the server procedure.

NOTE 1 – It is possible for a server procedure to also be a client procedure if it makes a call to another procedure in order to complete its desired function.

Procedures have the ability to exchange data between the client and server via the use of parameters (see 5.2). In addition, client and server procedures may also share data through the use of global data (see 5.2.2). In order for the parameters specified by the client procedure to be interpreted correctly, the parameters are required to be marshalled (see 5.2.3) to a base form for transmission that is shared by both the client and the server procedure. After the data has been transmitted, the server procedure must then unmarshall (see 5.2.3) the data from the base form into datatypes that are defined in the server language or in the language binding to ISO/IEC 11404 - Language-Independent Datatypes for that particular language.

NOTE 2 – An example of the process of marshalling and unmarshalling of parameters would be if a Pascal client procedure made a call to a Fortran server procedure passing a single character parameter by value. The Pascal “char” datatype would map to a LID character. In order to have the LID character be transmitted to the server procedure, the LID character is marshalled to an appropriate ASN.1 value, for example, which is a form that would be understood by both the client and server procedures. The ASN.1 value would then be transmitted to the server and upon receipt it is unmarshalled into a LID character, which in turn maps to a “character*1” in Fortran.

The following diagram outlines the basic components of the language-independent call model:



Language-Independent Procedure Call Model

This model illustrates how the client and server procedures communicate when their implementations conform to this International Standard. The virtual contract between the client procedure and server procedure is defined by the Interface Definition Notation contained within this International Standard. Upon the instantiation of a call, the marshalling interface marshalls the parameters and passes this information on to the client LIPC provider. The client LIPC provider is connected to the server LIPC provider via the actual contract which is the transmissible form (e.g., ASN.1). The server LIPC provider then unmarshalls the data, via the unmarshalling interface, into a form that is compatible with the server procedure. Upon return, the process is reversed with the unmarshalling interface now being the marshalling interface and the marshalling interface now being the unmarshalling interface.

5.2 Parameter passing

Any datatype defined in ISO/IEC 11404 - Language-Independent Datatypes can be the datatype of a formal parameter of a language-independent procedure call. This International Standard defines parameter passing solely on the passing of values. Therefore an actual parameter is any value of the datatype required by the call. The parameter passing model defined in this International Standard is a strongly typed model.

NOTE 1 – Weak typing can be accomplished by relaxing association rules and adding implicit datatype conversions in the language bindings to this International Standard.

The following notes relate the common parameter passing mechanisms that are found in existing languages to the four defined parameter passing schemes that are defined in this International Standard.

NOTES

- 2 **Call by Value (In parameters):** This is the simplest of all common parameter passing mechanisms and appears directly in LIPC as Call by Value Sent on Initiation (see 5.2.1.1). The virtual contract is fulfilled by the client evaluating the actual parameter and sending the value to the server procedure, and the server procedure accepting it. No further action is required of the client procedure. The server procedure does what it likes with the received value, but can make no further demands on the client with respect to the actual parameter that generated the value.
- 3 **Call by Value Return (Out parameters):** This common parameter passing mechanism is also directly supported in LIPC by Call by Value Returned as Specified. The virtual contract for this mechanism involves the concept of passing a parameter only as a means of receiving a value. If in a specific language binding, a parameter is passed at the language processor level, what is passed is an implicit pointer to a value of the datatype concerned, which the server procedure contracts to set. The server procedure cannot access the value of the datatype prior to the call. Some languages, in their datatyping model, explicitly distinguish between the datatypes of values held by variables and those of the variables themselves. For example, some languages have an explicit dereference (i.e., obtain the value of). For languages without such a model, the LIPC model allows that distinction to be made at the language binding service contract level without disturbing the virtual contract model.
- 4 **Call by Value Sent and Return (In-out parameters):** This common parameter passing mechanism is an in/out mechanism where the actual parameter can be evaluated to a destination for Call by Value Returned on Termination (see 5.2.1.3). However, in the LIPC model it is regarded as a parameter with both that property and that of Call by Value Sent on Initiation (see 5.2.1.1). Equivalently, it can be expanded into two implicit parameters, one of each kind. The actual parameter corresponding to a formal parameter of a given datatype "t" must be capable, on evaluation, of yielding a destination for such a value (i.e., an implicit or explicit pointer to a value of datatype "t"). For the "in" part of the in/out specification, the current value held in that destination on initiation of the call is retrieved by the client and relayed to the server procedure. The destination itself is also recorded. In the virtual contract the client receives the returned value, the "out" part of the in/out specification, from the server procedure and sends it to that destination.
Where the language binding or service contract passes the destination itself to the server procedure as part of the copy-in/copy-out, the server procedure must contract to retrieve the "in" value immediately on transfer and then to send the returned "out" value to the destination on completion of the call. While the call is in progress, the client explicitly or implicitly marks the destination as 'read once only, write once only' as far as the server procedure is concerned and any attempt by the server procedure to violate that condition is an error.
- 5 **Call by Reference:** In this case a formal parameter of datatype "t" is interpreted as an implicit 'pointer to "t"' and the actual parameter must evaluate to such a pointer accordingly. This pointer to "t" is then passed by value as an "in" parameter.
The pointer is not passed as an in/out parameter since this would cause an extra level of indirect addressing.
The virtual contract is that the client provides an access path to the destination. The destination is fixed, but the access path can be used by the server procedure both reading and writing of values of datatype "t". In the close-coupled case the service contract may well involve passing the actual destination with the client needing to take no further action until the call is complete. In a loosely-coupled service environment the service contract will involve client action during the call, responding to requests by the server for a value of datatype "t" to be read or written. In effect this would be reciprocal calls with the "in" and "out" directions reversed.
These reciprocal calls implied by Call by Reference in a loosely-coupled environment represent a potentially significant overhead, which may result in Call by Reference not being supported

in such services.

5.2.1 Methods of parameter passing

There are four basic kinds of parameter passing defined in this International Standard:

1. Call by Value Sent on Initiation
2. Call by Value Sent on Request
3. Call by Value Returned on Termination
4. Call by Value Returned when Available

5.2.1.1 Call by Value Sent on Initiation

This is the simplest form of parameter passing. The formal parameter of the server procedure receives a value of the datatype concerned. The virtual contract is that the client evaluates the actual parameter and supplies the resulting value at the time of transfer of control. The server procedure accepts this value and no further interaction takes place with respect to this parameter.

NOTE – This type of parameter passing is commonly known as Call by Value.

5.2.1.2 Call by Value Sent on Request

The virtual contract for this type of parameter passing is that the client undertakes to evaluate the actual parameter and supply the resulting value, but only upon receipt of a request to do so from the server procedure. The evaluation and passing of the actual parameter takes place if and only if the server procedure requests it. This can be done at the beginning of the call, or while the call is in progress.

The essential difference from Call by Value Sent on Initiation is that in some cases the value sent will be different.

NOTES

- 1 While this mechanism is not common to programming languages as an explicit standards requirement, it is an optimization mechanism for programming language implementations.
- 2 An example of the use of Call by Value Sent on Request is when a client wishes the server procedure to record a time, and wishes that to be done at a specific point during the execution of the call, rather than at the initiation of the call.
- 3 The use by the server of a parameter of the Call by Value Sent on Request type can be regarded as a call of an implicit procedure parameter where the server procedure does the evaluation one time. Any further reference in the server procedure to the formal parameter simply uses that same value. The server procedure does not issue a further request for a value.

5.2.1.3 Call by Value Returned on Termination

In this type of parameter passing, the virtual contract is that at the completion of the call, the server procedure will supply a value of the datatype of the formal parameter and the client will accept it and send the returned value to the appropriate destination.

NOTE 1 – This type of parameter passing is better known as Call by Value Return and is essentially the “out” equivalent of Call by Value Sent on Initiation.

Conceptually the client and not the server procedure sends the returned value to the destination, because the client language or mapping determines the interpretation of the destination and the process of return.

NOTES

- 2 In a closely coupled environment where providing the actual destination (perhaps even the hardware address) to the server procedure is a trivial task, there is no reason why the actual service contract at the implementation level should not include providing the actual destination to the server procedure, which then sends its returned value directly there. This is an additional service level function that the server procedure contracts to perform for the client procedure, which does not affect the logical division of responsibility at the virtual contract level.
- 3 This kind of parameter passing also accommodates the return of a value for the procedure as a whole, in the case of function procedures. Parameter passing utilizing Call by Value Returned on Termination accommodates function procedures through the use of an additional anonymous parameter.

5.2.1.4 Call by Value Returned when Available

In this type of parameter passing, the server procedure returns the parameter value at any time after the returned value is available. It could be returned while the call is still in progress, at the completion of the call, or some time later. What time is chosen is determined by the binding of the LIPC based service and is not defined by this International Standard. All the LIPC model requires is that this possibility can be accommodated. The virtual contract is that whenever the server procedure returns the value, the client will accept it and send the returned value to the appropriate destination.

NOTE – In this type of parameter passing, the possibility that the returned value will be returned more than once is not excluded.

5.2.2 Global data

The term *global data* is used for data defined in a shared execution context that can be referenced by another procedure executing in a different invocation context within the same execution context. Conceptually, global data requires the marshalling/unmarshalling of global data into individual information units. Implementations conforming to this International Standard may support an implementation-defined mechanism for the sharing of global data and may support partitioning of global data. Partitioning of data refers to the ability to insulate data from a procedure. It is

recommended that implementations support global data via implicit parameters that are passed on the call, but this may not be the only valid mechanism where the marshalling/unmarshalling operations are known to be trivial.

NOTES

- 1 In the IDN, global data is represented as an explicit parameter to the procedure. In a particular language mapping, these explicit parameters can be provided to the procedure using such mechanisms as external variables and as such are implicit parameters.
- 2 Global data should be available to the server by the time it is needed (i.e., before invocation, at invocation, before use is required, or at the time access is required).
- 3 The mechanism by which objects in the invocation context are associated to the global objects may be defined by the language, language mapping, or left to the implementation.

5.2.3 Parameter Marshalling / Unmarshalling

Data which is communicated between the client and server procedure needs to be assembled in a transmissible form. This transmissible form will allow the client and server procedures to encode their LID mapped data into a form that is suitable for both language-independent calling on the same system and remote procedure calls. The specification of this transmissible form is outside the scope of this International Standard.

NOTE – The Abstract Syntax Notation - One is a suitable specification of a transmissible form.

The marshalling of data refers to what the client procedure must do in order to transform its data into a form for transmission to the server procedure. Unmarshalling of data refers to what the server procedure must do in order to take the data passed by the client procedure and transform this into data suitable for the language of the server procedure. Marshalling is not limited to calling a procedure. Upon return, the server procedure must marshal any returned data into the form shared by the two procedures. Unmarshalling of data is not limited to the server procedure, since the client procedure must be able to unmarshal any data that is returned by the server procedure.

Since marshalling and unmarshalling of data for procedure calls is often complex and degrades performance, an implementation may want to perform optimization of this process wherever possible. Optimizations will likely be available when the client and server systems are homogeneous and the languages involved in the procedure call have the same data representation.

5.2.4 Pointer Parameters

A Call by Value Sent on Initiation of a pointer allows access to the entity pointed to. The pointer value itself cannot be changed by the server procedure in order for the pointer to refer to something else after the call.

NOTE – For example, if the value sent is a pointer to a record, after the call the pointer still points to the same record even though the values in the fields of the record may have changed.

If changing what the pointer refers to is needed, then another level of indirect referencing has to be invoked, either directly (as with call by reference) or indirectly (as with call by value-return). An access path via pointer parameters implies access to all lower levels, including the primitive datatype values referenced by the lowest level pointers.

5.2.5 Private types

A private type is a datatype that is protected from modification within the server procedure regardless of the attributes on a parameter being passed as a private type. No operations shall be permitted on a parameter of a private type. A private type is declared by including the restricted keyword prior to the LID datatype in the IDN.

NOTE – A private type can be considered as an octet stream that can have no operations performed on it.

5.3 Execution-time Control

5.3.1 Terminations

An implementation conforming to this International Standard shall provide a method for raising and handling terminations that occur during the initialization, execution, or completion of a procedure call. Raising a termination does not necessarily imply that the server procedure should be terminated immediately, however terminations that are raised must not be ignored by the implementation. Some examples of possible terminations include:

- normal termination of a procedure invocation returning the output parameters, input/output parameters, and result (if any)
- abnormal termination, in which the procedure itself detects an error or other unusual condition
- external cancellation, in which some other entity determines that the procedure should terminate
- hardware or software detected events which may or may not be critical to the proper execution of the application
- asynchronous events or notification
- type or value mismatches in parameter passing or return
- failure of the underlying invocation service itself.

5.3.1.1 Normal termination

A procedure completing normally raises a termination signifying a normal return. A procedure may report additional terminations; e.g., at return from a synchronous procedure call, the procedure may return two or more terminations; however, the first of these terminations must specify whether termination is normal, abnormal, or via a cancel. If the procedure call is asynchronous, the procedure may return an additional termination code before, during, or after termination.

5.3.1.2 Abnormal Termination

A procedure completing abnormally raises a termination as a result of some condition other than an external cancel command. The usual reason a procedure abnormally terminates is that the procedure encounters some condition that makes it impossible to continue or impossible to complete successfully the function(s) requested by the client procedure. The client procedure is notified by the implementation defined termination raising mechanism. Abnormal terminations can be divided into two cases:

- a procedure detects an abnormal termination as part of its logic and executes an explicit abort procedure as a result
- an abnormal termination occurs during execution of the procedure, causing a fault at some level lower than that of the procedure logic; the fault causes control to go to some generic fault-handling routine within the procedure that terminates the procedure as in the previous case.

A special case of abnormal termination of a procedure is the case where one or more of the actual parameter values in the procedure call are incorrect, e.g., a value is of the wrong datatype for a given parameter or of the right datatype but outside the required range. It is possible to distinguish here between parameter values that violate the advertised requirements of the procedure interface as specified in the IDN and values (or combination of values) that violate application specific constraints that cannot be specified in the IDN formalism and hence must be checked explicitly by the procedure itself. However, from the point of view of the client procedure, the only difference between the two cases is that in the first case, the error specified is one of a predefined set specified in this International Standard (see 5.3.1.4). In the second case, it is an application-specific condition code specified in some other, perhaps application-specific, standard.

5.3.1.3 External Cancellation

A procedure terminates by external cancellation if a command is issued from outside the procedure which causes the procedure to terminate, or be terminated, in an orderly way. In the case of an asynchronous call, the cancellation may come from the client procedure. Whether the call is synchronous or asynchronous, the command to cancel a procedure may come from an outside source, i.e., outside the LIPC model. The two cases are indistinguishable to the server procedure. In both cases, the client procedure receives a notification via an implementation defined termination raising mechanism.

5.3.1.4 Predefined conditions

As a minimum, implementations conforming to this International Standard should report the following terminations during a procedure call:

- server procedure unavailable, call not executed
- client or server procedure does not have defined mapping to IDN

- value out of range for parameter datatype
- cancellation of call
- insufficient resources available to complete call
- normal completion of call

5.4 Execution Control

5.4.1 Synchronous and Asynchronous Calling

The issue of whether of not a call executes synchronously or asynchronously is outside the scope of this International Standard. The LIPC model does not prohibit either synchronous or asynchronous calls. An implementation can choose whether or not to limit the number of threads of execution in any particular call environment.

5.4.2 Recursion

The LIPC model does not prohibit recursion. How an implementation implements recursive procedure calling is outside the scope of this International Standard.

NOTE – Implementors should be aware that optimization considerations for LIPC calls need to take recursion into account.

6 A model of procedure calling: formal description

This clause provides a model of procedures, variables, name bindings, execution environments, and invocation. A series of new datatypes are introduced. Some of these directly correspond to programming concepts (like variables), and some are used merely to support further definitions.

6.1 Value

The set Value contains all the values that might arise in a program execution. Value contains all the values definable using the datatypes, type generators, and definitional mechanisms of ISO/IEC 11404 - Language-Independent Datatypes. Value will also contain boxes and procedure closures (see 6.6).

6.2 Boxes and global state

A box is a generic term for a container that holds a value of a particular datatype, for example what, in some contexts, would be called a 'variable'. Boxes exist and are manipulated at execution-time. They may be named by identifiers in some program text, but they are distinct from any such syntactic notion. Boxes do not imply any particular implementation mechanism such as storage. There are three operations defined on boxes: