INTERNATIONAL STANDARD

ISO/IEC 13719-1

First edition
1995-06-01

Information technology — Portable Common Tool Environment (PCTE) —

Part 1:
Abstract specification

Technologies de l'information — Environnement d'outil courant portable (PCTE) —

Partie 1: Spécification d'abstrait

Reference number
ISO/IEC 13719-1:1995(E)

Documents
Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

Contents

1 Scope 1

2 Conformance 1

2.1 Conformance of binding 1
2.2 Conformance of implementation 2

3 Normative references 3

4 Definitions 3

4.1 Technical terms 3
4.2 Other terms 4

5 Formal notations 4

6 Overview of PCTE 4

6.1 PCTE structural architecture 5
6.2 Object management system 5
6.3 Object base 5
6.4 Schema management 6
6.5 Self-representation and predefined SDSs 6
6.6 Object contents 6
6.7 Process execution 7
6.8 Monitoring 7
6.9 Communication between processes 7
6.10 Notification 7
6.11 Concurrency and integrity control 7

© ISO/IEC 1995
All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

Documents
Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

6.12 Distribution 8
6.13 Replication 8
6.14 Security 8
6.15 Accounting 9
6.16 Implementation limits 9

7 Outline of ISO/IEC 13719 9

8 Foundation 10

8.1 The state 10
8.2 The object base 11
8.2.1 Objects 11
8.2.2 Attributes 12
8.2.3 Links 13
8.3 Types 14
8.3.1 Object types 14
8.3.2 Attribute types 15
8.3.3 Link types 16
8.3.4 Enumeral types 19
8.4 Types in SDS 19
8.4.1 Object types in SDS 21
8.4.2 Attribute types in SDS 21
8.4.3 Link types in SDS 22
8.4.4 Enumeral types in SDS 22
8.5 Types in working schema 22
8.5.1 Object types in working schema 23
8.5.2 Attribute types in working schema 23
8.5.3 Link types in working schema 24
8.5.4 Enumeral types in working schema 24
8.6 Types in global schema 24
8.7 Operations 25
8.7.1 Calling process 25
8.7.2 Direct and indirect effects 25
8.7.3 Errors 27
8.7.4 Operation serializability 28

9 Object management 28

9.1 Object management concepts 28
9.1.1 The basic type "object" 28
9.1.2 The common root 31
9.1.3 Datatypes for object management 32
9.2 Link operations 32
9.3 Object operations 40
9.4 Version operations 53

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

10 Schema management **59**

10.1 Schema management concepts 59
10.1.1 Schema definition sets and the SDS directory 59
10.1.2 Types 60
10.1.3 Object types 61
10.1.4 Attribute types 61
10.1.5 Link types 63
10.1.6 Enumeral types 64
10.1.7 Datatypes for schema management 64

10.2 SDS update operations 64

10.3 SDS usage operations 89

10.4 Working schema operations 96

11 Volumes, devices, and archives **100**

11.1 Volume, device, and archiving concepts 100
11.1.1 Volumes 100
11.1.2 Administration volumes 101
11.1.3 Devices 102
11.1.4 Archives 102

11.2 Volume, device, and archive operations 103

12 Files, pipes, and devices **110**

12.1 File, pipe, and device concepts 110

12.2 File, pipe, and device operations 113

13 Process execution **120**

13.1 Process execution concepts 120
13.1.1 Static contexts 120
13.1.2 Foreign execution images 121
13.1.3 Execution classes 121
13.1.4 Processes 122
13.1.5 Initial processes 128
13.1.6 Profiling and monitoring concepts 129

13.2 Process execution operations 129

13.3 Security operations 141

13.4 Profiling operations 146

13.5 Monitoring operations 147

14 Message queues **149**

14.1 Message queue concepts 149

14.2 Message queue operations 151

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

15 Notification 157

15.1 Notification concepts 157
15.1.1 Access events and notifiers 157
15.1.2 Notification messages 158
15.1.3 Time of sending notification messages 158
15.1.4 Range of concerned message queues 158

15.2 Notification operations 159

16 Concurrency and integrity control 160

16.1 Concurrency and integrity control concepts 160
16.1.1 Activities 160
16.1.2 Resources and locks 162
16.1.3 Lock modes 164
16.1.4 Inheritance of locks 166
16.1.5 Establishment and promotion of locks 167
16.1.6 Implied locks 168
16.1.7 Conditions for establishment or promotion of a lock 169
16.1.8 Releasing locks 169
16.1.9 Permanence of updates 170
16.1.10 Tables for locks 170

16.2 Concurrency and integrity control operations 173

17 Replication 178

17.1 Replication concepts 178
17.1.1 Replica sets 178
17.1.2 Replicated objects 179
17.1.3 Selection of an appropriate replica 180
17.1.4 Administration replica set 181

17.2 Replication operations 181

18 Network connection 187

18.1 Network connection concepts 187
18.1.1 Execution sites 187
18.1.2 Workstations 187
18.1.3 Foreign systems 190
18.1.4 Network partitions 190
18.1.5 Accessibility 191
18.1.6 Workstation closedown 192

18.2 Network connection operations 193

18.3 Foreign system operations 198

18.4 Time operations 199

19 Discretionary security 200

19.1 Discretionary security concepts 200
19.1.1 Security groups 200
19.1.2 Access control lists 203

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

19.1.3 Discretionary access modes 206
19.1.4 Access control lists on object creation 208
19.2 Operations for discretionary access control operation 209
19.3 Discretionary security administration operations 212

20 Mandatory security 217

20.1 Mandatory security concepts 217
20.1.1 Mandatory classes 217
20.1.2 The mandatory class structure 218
20.1.3 Labels and the concept of dominance 219
20.1.4 Mandatory rules for information flow 220
20.1.5 Multi-level security labels 224
20.1.6 Floating security levels 226
20.1.7 Implementation restrictions 228
20.1.8 Built-in policy aspects 228
20.2 Operations for mandatory security operation 230
20.3 Mandatory security administration operations 235
20.4 Mandatory security operations for processes 239

21 Auditing 241

21.1 Auditing concepts 241
21.1.1 Audit files 241
21.1.2 Audit selection criteria 243
21.2 Auditing operations 244

22 Accounting 248

22.1 Accounting concepts 248
22.1.1 Consumers and accountable resources 248
22.1.2 Accounting logs and accounting records 249
22.2 Accounting administration operations 252
22.3 Consumer identity operations 257

23 Common binding features 257

23.1 Mapping of types 257
23.1.1 Mapping of predefined PCTE datatypes 257
23.1.2 Mapping of designators and nominators 260
23.1.3 Mapping of other values 266
23.2 Object reference operations 267
23.3 Link reference operations 269
23.4 Type reference operations 272

24 Implementation limits 274

24.1 Bounds on installation-wide limits 274

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

24.2 Bounds on workstation-dependent limits 275

Annex A (normative) **VDM Specification Lnaguage for the abstract specification** **278**

Annex B (normative) **The Data Definition Language (DDL)** **283**

Annex C (normative) **Specification of errors** **292**

Annex D (normative) **Auditable events** **311**

Annex E (informative) **The predefined schema definition sets** **318**

Index of Error Conditions **337**

Index of Technical Terms **344**

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

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 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. 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 13719-1 was prepared by the European Computer Manufacturers Association (ECMA) (as Standard ECMA-149) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

ISO/IEC 13719 consists of the following parts, under the general title *Information technology — Portable Common Tool Environment (PCTE)*:

— *Part 1: Abstract specification*

— *Part 2: C programming language binding*

— *Part 3: Ada programming language binding*

Annexes A to D form an integral part of this part of ISO/IEC. Annex E is for information only.

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

Information technology — Portable Common Tool Environment (PCTE) —

Part 1:
Abstract specification

1 Scope

This part of ISO/IEC 13719 specifies PCTE in abstract, programming-language-independent, terms. It specifies the interface supported by any conforming implementation as a set of abstract operation specifications, together with the types of their parameters and results. It is supported by a number of standard *bindings*, i.e. representations of the interface in standard programming languages.

The scope of this part of ISO/IEC 13719 is restricted to a single PCTE installation. It does not specify the means of communication between PCTE installations, nor between a PCTE installation and another system.

A number of features are not completely defined in this part of ISO/IEC 13719, some freedom being allowed to the implementor. Some of these are *implementation limits*, for which constraints are defined (see clause 24). The other implementation-dependent and implementation-defined features are specified in the appropriate places in this part of ISO/IEC 13719 .

PCTE is an interface to a set of facilities that forms the basis for constructing environments supporting systems engineering projects. These facilities are designed particularly to provide an infrastructure for programs which may be part of such environments. Such programs, which are used as aids to systems development, are often referred to as tools.

2 Conformance

2.1 Conformance of binding

A binding conforms to this part of ISO/IEC 13719 if and only if:

- it consists of a set of operational interfaces and datatypes, with a mapping from the operations and datatypes of this part of ISO/IEC 13719 ;
- each operation of this part of ISO/IEC 13719 is mapped to one or more sequences of one or more operations of the binding (distinct operations need not be mapped to distinct sets of sequences of binding operations);
- each datatype of this part of ISO/IEC 13719 is mapped to one or more datatypes of the binding;
- each named error of this part of ISO/IEC 13719 is mapped to one or more error values (status values, exceptions, or the like) of the binding;

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

- the conditions of clause 23 on common binding features are satisfied;
- the conditions for conformance of an implementation to the binding are defined, are achievable, and are not in conflict with the conditions in 2.2 below.

2.2 Conformance of implementation

The functionality of PCTE is divided into the following modules:

- The core module consists of the datatypes and operations defined in clauses 8 to 19 (except 13.1.6, 13.4, and 13.5) and 23.
- The mandatory access control module consists of the datatypes and operations defined in clause 20.
- The auditing module consists of the datatypes and operations defined in clause 21.
- The accounting module consists of the datatypes and operations defined in clause 22.
- The profiling module consists of the datatypes defined in 13.1.6 and the operations defined in 13.4.
- The monitoring module consists of the datatype Address defined in 13.1.6 and operations defined in 13.5.

An implementation of PCTE conforms to this part of ISO/IEC 13719 if and only if it implements the core module.

An implementation of PCTE conforms to this part of ISO/IEC 13719 with mandatory access control level 1 or 2 if it implements the core module and in addition:

- for level 1: the mandatory access control module except the floating security levels features defined in 20.1.6;
- for level 2: the mandatory access control module.

An implementation of PCTE conforms to this part of ISO/IEC 13719 with auditing if and only if it implements the core module and in addition the auditing module.

An implementation of PCTE conforms to this part of ISO/IEC 13719 with accounting if and only if it implements the core module and in addition the accounting module.

An implementation of PCTE conforms to this part of ISO/IEC 13719 with profiling if and only if it implements the core module and in addition the profiling module.

An implementation of PCTE conforms to this part of ISO/IEC 13719 with monitoring if and only if it implements the core module and in addition the monitoring module.

By 'an implementation implements a module' is meant that, for the clauses of the module:

- the implementation conforms to a binding of this part of ISO/IEC 13719 which itself conforms to this part of ISO/IEC 13719 and which is itself an International Standard;
- if an operation of this part of ISO/IEC 13719 is mapped to a set of sequences of operations in the binding:
  - . case 1: operation_A; operation_B; ... operation_F;
  - . case 2: operation_G; operation_H; ...operation_M;
  - . etc.

  then in each case the sequence of invocations of the operations of the implementation must have the effect of the original operation of this part of ISO/IEC 13719;
- the relevant limits on quantities specified in clause 24 are no more restrictive than the values specified there;

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

- the implementations of the implementation-defined features in this part of ISO/IEC 13719 are all defined.

An implementation of PCTE does not conform to this part of ISO/IEC 13719 if it implements any of the following, whether or not the PCTE entity mentioned is in a module which the implementation implements:

- an operation with same name as a PCTE operation but with different effect;
- an SDS with the same name as a PCTE predefined SDS but with different contents;
- an error condition with the same name as a PCTE error condition but with different meaning.

3 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 13719. 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 currently valid International Standards.

ISO/IEC 2022 : 1994,	*Information technology — Character code structure and extension techniques.*
ISO 8601 : 1988,	*Data elements and interchange formats — Information interchange — Representation of dates and times.*
ISO 8859-1 : 1987,	*Information processing — 8-bit single-byte coded graphic character sets — Part 1 : Latin alphabet No. 1.*
ISO/IEC 10646-1 : 1993,	*Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.*
ISO/IEC 11404 : —[1)],	*Information technology — Programming languages, their environments and system software interfaces — Language-independent datatypes.*
ISO/IEC 13303-1 : —[1)],	*Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method/Specification language — Part 1 : Basic Language.*
BS 6145 : 1981	*Method of Defining Syntactic Metalanguage.*

4 Definitions

4.1 Technical terms

All technical terms used in this part of ISO/IEC 13719, other than a few in widespread use, are defined in the text, usually in a formal notation. All identifiers defined in VDM-SL or in DDL (see 5.2) are technical terms; apart from those, a defined technical term is printed in italics at the point of its definition, and only there. For the use of technical terms defined in VDM-SL and DDL see clause A.3 and clause B.9 respectively. All defined technical terms are listed in an index, with references to their definitions.

1) To be published.

Documents Document Sharing Hub
Full document details: ISO_IEC 13719-1_1995

4.2 Other terms

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

4.2.1 implementation-defined: Possibly differing between PCTE implementations, but defined for any particular PCTE implementation.

4.2.2 implementation-dependent: Possibly differing between PCTE implementations and not necessarily defined for any particular PCTE implementation.

4.2.3 binding-defined: Possibly differing between language bindings, but defined for any particular language binding.

4.2.4 datatype: The type of a parameter or result of an operation defined in this part of ISO/IEC 13719, or used to define such a type. Where, as in clause 23, it is necessary to distinguish these types from datatypes defined elsewhere, the term *PCTE datatype* is used.

5 Formal notations

Four formal notations are used in this part of ISO/IEC 13719.

For datatypes and for operation signatures, a small subset of the *Vienna Development Method Specification Language* or *VDM-SL* is used; it is defined in annex A. This subset of VDM-SL is also used to define some types used for operation parameters and results.

The *Data Definition Language* or *DDL* is used to define types; it is defined in annex B. Where a concept is defined in both VDM-SL and DDL, the same identifier is used.

To define the error conditions detected by operations, a parameterized notation is used; it is defined in annex C.

The BSI syntactic notation (BS 6154 : 1981) is used to define the syntax of VDM-SL and DDL, and in a few other places where the syntax of strings is defined.

6 Overview of PCTE

PCTE is designed to support program portability by providing machine-independent access to a set of facilities. These facilities, which are described in ISO/IEC 13719, are designed particularly to provide an infrastructure for programs to support systems engineering projects.

The PCTE architecture is described in two dimensions: the *structural architecture* and the *functional architecture*. The structural architecture is described in 6.1, and shows how a PCTE installation is built of a system of communicating workstations and how the software providing the PCTE interfaces is structured. The functional architecture is described in 6.2 onwards, and gives an outline of the functional components of PCTE and the facilities they provide.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

6.1 PCTE structural architecture

The preferred structural architecture for a PCTE installation is a set of workstations and associated resources communicating over a network, though other architectures are possible. There is no hierarchy or ordering of workstations within a PCTE installation. If a workstation is part of a PCTE installation then the PCTE installation appears to the workstation's user as a conceptually single machine, although each workstation can act as an autonomous unit. Such a user has access to the total resources of a PCTE installation, subject to the necessary access controls.

The PCTE database (called the *object base*) is partitioned into volumes. Volumes are dynamically allocated to (*mounted on*) particular workstations, and, once mounted, are globally available in that PCTE installation.

The program writer does not need to be aware of the distribution architecture, but the PCTE interfaces do provide all the facilities needed to configure a PCTE installation and control its distribution. The PCTE interfaces appear to the tool writer as available within a PCTE installation irrespective of the tool's physical location within a PCTE installation and independent of any particular network topology.

6.2 Object management system

An aspect of PCTE that is of major importance to the process of constructing and integrating portable tools is the provision of the object base and a set of functions to manipulate the various objects in the object base. The object base is the repository of the data used by the tools of a PCTE installation, and the *Object Management System* or *OMS* of PCTE provides the functions used to access the object base.

In a general sense, the users and programs of the PCTE installation have the ability to manage entities that are known to, and can be designated in, a particular PCTE installation. These may be files in the traditional sense, or peripherals, interprocess message queues or pipes, or the description of processes themselves or of the static context of a process. Tools supporting user applications establish classes of objects defined by the user: these can represent information items such as project milestones, tasks, and change requests.

6.3 Object base

The basic OMS model is derived from the Entity Relationship data model and defines *objects* and *links* as being the basic items of a PCTE object base.

Objects are entities (in the Entity Relationship sense) which can be designated, and can optionally have:

- *Contents*: a storage of data representing the traditional file concept;
- *Attributes*: primitive values representing specific properties of an object which can be named individually;
- *Links*: representations of associations between objects. Links may have attributes, which may be used to describe properties of the associations or as keys to distinguish between links of the same type from the same object.

Designation of links is the basis for the designation of objects: the principal means for accessing objects in most OMS operations is to navigate the object base by traversing a sequence of links.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

6.4 Schema management

Entities used by the user and those used by the system that are represented by objects in the object base can be treated in a uniform manner, and facilities to control their structure, to store and to designate these objects, are provided by PCTE.

The object base of each PCTE installation is governed by a typing mechanism. All entities in the object base are typed and the data must conform to the corresponding type rules. Type rules are defined for objects, for links, and for attributes.

PCTE is designed to allow, but not to require, distributed and devolved management of the object base. To this end the definition of the typing rules which govern an object, a link, or an attribute in the object base may be split up among a number of *schema definition sets* (or *SDSs*). Some properties of an object, a link, or an attribute must be the same in every SDS which contributes to the definition of the typing rules for that object, link, or attribute: these are properties of the *type*. Other properties may differ for different SDSs: these are properties of the *type in SDS*.

Each SDS provides a consistent and self-contained view of the data in the object base. A process, at any one time, views the data in the object base through a *working schema*. A working schema is obtained as a composition of SDSs in an ordered list. The effect of such a composition is to provide a union of all the types contained in the listed SDSs. A uniform naming algorithm, dependent on the ordering of the SDSs, is applied to all the contained types.

The object base of a PCTE installation has a notional *global schema*, composed of all the SDSs. The global schema is not directly represented in the object base, and the concept is used mainly to state certain consistency constraints on the object base as a whole.

Child types of object types can be defined with the effect of implicit inheritance of all properties of their parent types. Additionally, child types can have properties of their own.

6.5 Self-representation and predefined SDSs

Many of the entities in a PCTE installation are represented by objects in the object base. The types of these objects are defined in *predefined SDSs*, which are available in any conforming implementation; for example processes are represented by objects of type "process" which is defined in the predefined SDS 'system'. This property of PCTE is called *self-representation*. In general, in this part of ISO/IEC 13719, the name of an entity is used also to refer to the object that represents it.

In some cases an object of a type representing some kind of entity requires initializing, or must be created by a particular operation, before it can be used in operations to represent an entity of that kind. Such an object which has been initialized or correctly created is referred to as a *known* entity of that kind (i.e. known to the PCTE installation); any other object of that type is referred to as an *unknown* entity. For example an object of type "process" created by PROCESS_CREATE is a known process, while one created by OBJECT_CREATE is an unknown process.

6.6 Object contents

A set of operations is provided to access the contents of some types of objects (files, pipes, and devices). These operations provide conventional input-output facilities on files and pipes and control of input and output on devices. These contents are not interpreted by PCTE.

Other types of objects (accounting logs and audit files) have contents with structure that is defined by PCTE and for access to which special operations are provided.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

6.7 Process execution

PCTE is an interface to support programs. When a program is *run*, this is either the *execution* of the program itself, or the execution of an interpreter which interprets the program. An execution of a program is a *process*. Processes are represented by objects in the object base, so the hierarchy of processes, the environment in which a process runs, the parameters it has been passed, and the various stages of the program execution can be controlled, manipulated and examined.

These facilities can be used also to control processes running on *foreign systems*. A *foreign system* can be a foreign development system, a target system running a real-time operating system, or even a PCTE workstation in another PCTE installation.

6.8 Monitoring

PCTE provides three sets of features to support debugging and monitoring of processes.

- To measure the amount of time spent in selected parts of the code.
- To observe, and modify, the execution of a child process.
- To measure the processor usage of the calling process.

6.9 Communication between processes

PCTE provides a number of different mechanisms for communicating between processes. The principal ones supplied are:

- the objects, links and attributes in the database;
- message queues;
- pipes.

Message queues and pipes are essentially special forms of object. Thus both pipes and message queues are special cases of the general use of the object base for *interprocess communication.*

Pipes and message queues also provide communication between PCTE processes and foreign processes running on foreign systems (if the foreign systems allow it).

6.10 Notification

In PCTE there is a mechanism that allows the designation of objects so that certain types of access result in a message being posted in a message queue which can be accessed by the process requesting the notification.

The notification mechanism allows a process to specify events, corresponding to operations on objects, of which it wants to be notified.

6.11 Concurrency and integrity control

The object base is subject to concurrent access by users, and is liable to underlying system failure.

PCTE provides locking facilities to control the strength of object base concurrency and consistency, ranging from unprotected behaviour, through protected behaviour, to protected atomic and serializable transaction activities. PCTE ensures object base consistency and object base integrity for atomic and serializable transactions.

Each user carrying out a transaction on the object base sees some grouping of operations as an atomic operation which transforms the object base from one consistent state to another. If transactions are run one at a time then each transaction sees the consistent state left by its

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

predecessor. When transactions are run concurrently PCTE ensures that the effect on the object base is as though they were run serially. With a few exceptions, such as messages sent to or received from a message queue, the effect of a sequence of operations performed within a transaction is atomic: either all the operations are performed or none are performed.

Another important aspect of activities arises in composition of programs. A single program carrying out an atomic transaction on the object base can be regarded as performing a single function. More powerful functions can be built up by an outer program invoking a set of other, inner, programs, each of which carries out its own specific function. PCTE provides *nested activities* to allow each inner activity to behave in an atomic way, and at the same time to allow the whole function to be atomic. Thus the outer program can start a transaction, which may be either committed or aborted, and finally the whole outer transaction is committed or aborted. Each such inner program could itself invoke further nested programs, and so on.

6.12 Distribution

PCTE is based on a community of workstations of possibly differing types connected together by a network. The community is normally seen by the user as a single environment, grouping together the facilities, services and resources of all the different workstations, though in some circumstances a PCTE installation may be temporarily divided into separated partitions, each of which supports useful work.

Objects, including processes, are distributed throughout a PCTE installation. A user is able to disregard both the location of objects on volumes in the network and that of the workstation concerned in executing processes. Alternatively a user may choose to exercise control over the location of objects on volumes and the location of processes. On creation of an object a volume can be specified to indicate its location. Every process executes on a particular workstation and a user can specify which workstation by either static or dynamic means: the static context of a program has an execution class identifying the range of workstations upon which the static context may be executed; the workstation on which a process executes can be specified on invocation.

6.13 Replication

As it is possible that one or more workstations of a PCTE installation become temporarily unavailable, certain installation-wide objects must still be accessible. Replication facilities are available whereby a copy of an object's contents, attributes and links are made to each workstation. Installation-wide objects are predefined as replicated and other objects can be added. This feature is intended for non-volatile, rarely varying, widely consulted objects.

6.14 Security

A PCTE installation has to support many users and many projects. Different users are expected to have different roles within projects and to be authorized to access different objects. The user accesses objects using programs (themselves modelled as static contexts within the object base).

The purpose of security is to prevent the unauthorized disclosure, amendment or deletion of information. Security facilities are provided to support the definition of the different authorizations of users and programs.

Security in PCTE is provided by discretionary and mandatory access controls. Access controls as defined in the security clauses form one aspect of the correct operation of the installation with regard to the integrity of the information held and the correctness of its use. In this regard, the facilities described in the security clauses complement the data modelling facilities of the OMS and schema management, and the transaction and concurrency control facilities.

Each OMS object is associated with *access control lists* which define which types of access to the object are permitted for designated users or programs. Access control lists are expressed in terms

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

of *discretionary access rights* which are explicitly granted or denied to designated individual users, user groups or program groups. Access rights on a particular object are combined in order to determine a process's permission to perform each particular operation on the object.

Mandatory access controls cover both *mandatory confidentiality* and *mandatory integrity*, with distinct controls. Mandatory access controls are additional to discretionary access controls.

Mandatory confidentiality controls prevent the disclosure of information to unauthorized users. They prevent the flow of information to the unauthorized user directly, by controlling read access (*simple confidentiality*), and indirectly, by controlling the flow of information between objects (*confidentiality confinement*).

Mandatory integrity controls prevent unauthorized sources from contributing to the information in an object. They prevent the flow of information from the unauthorized user directly, by controlling write access (*simple integrity*), and indirectly, by controlling the flow of information between objects (*integrity confinement*).

6.15 Accounting

The accounting facilities of PCTE allow the automatic recording of the consumption of selected installation resources by users, groups of users, or groups of programs.

Authorized users may designate selected objects like programs, files, pipes, message queues, devices, workstations, and SDSs as being accountable resources. Access to an accountable resource by a process implies the automatic logging of usage information into the associated accounting log on completion of the operation.

6.16 Implementation limits

PCTE permits the user to examine the implementation-defined limits for the PCTE installation in which a program executes.

Minimal values are defined for limits, so that a program respecting those values is portable to any PCTE installation.

7 Outline of ISO/IEC 13719

Clause 6 gives an informal, non-normative explanation of the concepts of PCTE. Clause 7 gives an overview of the document and of the structure of the definition.

The partly formal, normative definition of PCTE is in clauses 8 to 24 and annexes A to C. It is in two main parts. The first main part is the *foundation* (clause 8) which defines the concept Object and its parts, for example Attribute and Link, and the concepts of the associated typing mechanism, for example Type and Type in SDS. This uses a subset of VDM-SL; see annex A.

The second main part of the definition is the interface definition (clauses 9-22). This defines the other concepts of PCTE, for example Process and Workstation, as specializations of the concept Object (clauses 11-22). This definition is in terms of the typing structure associated with these specializations, that is in terms of the typing concepts of the foundation. A language for the definition of types and types in SDS, called *Data Definition Language* or *DDL*, is defined in annex B.

The concept Object is itself further specialized, i.e. details not necessary for the foundation are added, in clause 9. (The name *Object* is used in both the foundation and the interface definition because it is the same concept although only a few of its details are defined in the foundation.)

Thus the foundation is a relatively simple general model that is specialized in later clauses to provide the PCTE interface definition.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

Instances of the PCTE concepts are called *entities* and they are referred to by the names of the underlying concepts, for example instances of Object are called objects. All the entities existing at a time are called the *state* of the PCTE installation. PCTE is defined in terms of the permissible values of the state and the permissible operations on the state. The foundation defines part of the state, namely that part concerned with entities of the foundation concepts; the interface definition defines the rest of the state and all the operations.

The concepts of the typing mechanism cannot be treated as specializations of the concept Object because the definition of PCTE would then be circular. They can however be *represented* by specializations of Object so that tools can determine the current state of the typing mechanism using the operations provided for determining the current state of objects. Operations for manipulating the state of the typing mechanism also manipulate the representing objects automatically and equivalently. The representations and operations of the typing mechanism are defined in clause 10.

The interface is defined by operations grouped according to function. For each group some concepts are defined first in DDL and possibly VDM-SL, as described above. There follow the operation definitions; a VDM-SL definition of the signature, an informal English description of the normal action of the operation, and a list of the possible error conditions (using an abbreviated notation defined in annex C).

Other parts of ISO/IEC 13719 define application programming interfaces to PCTE in terms of specific programming languages by defining the mapping of datatypes, operations, and error conditions of the abstract specification to datatypes, operations, and error conditions respectively of the programming language (see 3.1). Such mapping specifications are called *bindings*. Clause 23 defines a number of features to which all bindings must conform.

Clause 24 defines the limits on the sizes and numbers of various entities which a conforming PCTE implementation must respect. These are given as minima which an implementation must meet or exceed.

Annexes A to C define various notations used in the Abstract Specification. Annex A defines the subset of VDM-SL used for type definitions and operation signatures; annex B defines DDL; and annex C defines the notation for operation error conditions.

Annex D is provided for information; it collects the DDL definitions of the types in the predefined schema definition sets.

Annex E contains a list of auditable events classified by event type.

Annex F is provided for information; it contains an index of error conditions.

Clauses 8 to 24 contain commentary (headed NOTE or NOTES) which is not normative and is intended as a help to the reader in understanding the definition.

8 Foundation

8.1 The state

```
state PCTE_Installation of
    SYSTEM_TIME          : Time
    OBJECT_BASE          : map Object_designator to Object
    PROCESSES            : set of Process
    MESSAGE_QUEUES       : set of Message_queue
    CONTENTS_HANDLES     : map Contents_handle to Current_position
    CURRENT_POSITIONS    : map Current_position to Natural
    WORKSTATIONS         : set of Workstation
end

Name = Text

Name_sequence = seq of Name
```


AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

```
Working_schema ::
    VISIBLE_TYPES    : set of Type_in_working_schema
    SDS_NAMES        : Name_sequence

Process ::
    PROCESS_OBJECT   : Object_designator
    WORKING_SCHEMA   : Working_schema
    OPEN_CONTENTS    : set of Open_contents

Message_queue ::
    QUEUE_OBJECT  : Object_designator
    MESSAGES      : seq of Message

Workstation ::
    WORKSTATION_OBJECT  : Object_designator
    AUDIT_CRITERIA      : set of Selection_criterion
```

The state comprises the entities of a PCTE installation that endure from one operation call to another. The effect of an operation call is to modify the state, or to return values derived from the state (and any parameters), or both.

The system time is the date and time of day at any instant, as given by some system clock. For the format of the time see 23.1.1.5. The *current time* for an operation is a value of the system time at some moment between the start and end of the operation.

The object base is a set of objects identified by object designators (see 8.2.1).

A working schema is associated with a process (see clause 13) and consists of a set of types in working schema, derived from a sequence of SDSs. The types in working schema in the working schema of the calling process are called *visible types*. For the creation of a working schema for a process see 13.2.12.

The initial value of the state consists of the following objects:

- at least one workstation, at least one device managed by that workstation, at least one volume mounted on that device, and at least one process running on that workstation (see 18.1.2, 11.1.3, 11.1.1, and 13.1.5);
- the administration replica set, the common root, and the administrative objects (see 17.1.4 and 9.1.2);
- at least one user (see 19.1.1);
- at least the schema definition sets system, metasds, discretionary_security, mandatory_security (if implemented), and accounting (if implemented) (see 10.1);
- the predefined user group ALL_USERS, and the predefined program groups PCTE_AUDIT, PCTE_REPLICATION, PCTE_EXECUTION, PCTE_SECURITY, PCTE_HISTORY, PCTE_CONFIGURATION, and PCTE_SCHEMA_UPDATE (see 19.1.1).

NOTE - It is intended that the system time should be as near as possible the same throughout a PCTE installation.

8.2 The object base

8.2.1 Objects

```
Object ::
    OBJECT_TYPE              : Object_type_nominator
    ATTRIBUTES               : set of Attribute
    LINKS                    : set of Link
    DIRECT_COMPONENTS        : set of Object
    PREFERRED_LINK_TYPE      : [ Link_type_nominator ]
    PREFERRED_LINK_KEY       : [ Text ]
    CONTENTS                 : [ Contents ]
```

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

```
Object_designator :: Token
Object_designators = set of Object_designator
Contents = Structured_contents | Unstructured_contents
Structured_contents = Accounting_log | Audit_file
Unstructured_contents = File | Pipe | Device
Object_scope = ATOMIC | COMPOSITE
```

The object type constrains the properties of the object (see 8.3.1).

No two attributes of an object have the same attribute type. There is a basic set of attributes which all objects have; it is defined in 9.1.1.

The preferred link type and preferred link key, if present, are used as defaults in the identification of a link of the object (see 8.2.3). The preferred link key has the syntax of a key (see 23.1.2.7).

Every direct component of an object is the destination of a composition link of the object, and vice versa.

An *outer object* of an object A is an object of which A is a component.

The *atomic object* associated with an object comprises the links, attributes, preferred link type, preferred link key, and contents of the object. The *atoms* of an object are the atomic objects associated with the object and all its components.

A *component* of an object is a direct component of the object or of a component of the object. An object which is a component of each of two distinct objects, neither of which is a component of the other, is called a *shared component* of those two objects.

An *internal link* of an object is a link of the object or of one of its components for which the destination is either a component of the object or the object itself. An *external link* of an object is a direct or indirect outgoing link of the object which is not an internal link of the object. An object is called the *origin* of each of its links.

An object is specified by an object designator, or by a specialization of object designator defined as follows: if "X" is an object type (that is, it is a descendant of "system-object", see 9.1.1) then 'X_designator' (with capital initial) stands for 'Object_designator' with the condition that the value must designate an object of type "X" or a descendant of "X". For the mapping of object designators to the language bindings, see 23.1.2.2.

An object scope is used to indicate whether the effect of an operation applies to an object (COMPOSITE) or to the atomic object of the object (ATOMIC).

NOTES

1 An object can be a component of itself. Similarly two objects can be components of each other; in that case there are two distinct objects with the same atoms.

2 General operations are provided for handling unstructured contents (see clause 12) as a sequence of octets, the meaning of which is not further defined in this part of ISO/IEC 13719. Specific operations are provided for handling structured contents, which has a defined meaning in each case (see clauses 21 and 22).

3 When an object is created, so are all its attributes in the global schema. When a new attribute type is applied to the object's type in an SDS, effectively all objects of that type and its descendants gain a new attribute with its initial value. If the application of an attribute type to that object type is removed from all SDSs, the attribute remains on each object of that type until deleted by OBJECT_DELETE_ATTRIBUTE.

8.2.2 Attributes

```
Attribute ::
    ATTRIBUTE_TYPE     : Attribute_type_nominator
    ATTRIBUTE_VALUE    : Attribute_value
```


Documents
Document Sharing Hub
Full document details: ISO_IEC 13719-1_1995

Attribute_value = Integer | Natural | Boolean | Time | Float | String

Attribute_designator :: Token

Attribute_designators = **set of** Attribute_designator

Attribute_selection = Attribute_type_nominators | VISIBLE_ATTRIBUTE_TYPES

Attribute_assignments = **map** Attribute_designator **to** Attribute_value

String = **seq of** Octet

The value of an enumeration attribute is represented by its position within the enumeration value type (see 8.3.2).

An attribute is specified as follows:

- for an attribute of an object: an object designator which specifies the object, and an attribute designator which specifies the attribute relative to the object;
- for an attribute of a link: an object designator which specifies the origin of the link, a link designator which specifies the link relative to the object (see 8.2.3), and an attribute designator which specifies the attribute relative to the link.

NOTES

1 Each attribute in the object base is a key or non-key attribute of a link in the object base or a direct attribute of an object in the object base.

2 An implementation may impose constraints on the values of attributes (see clause 24). An attribute may take any value of its value type within those constraints; for example, a string attribute may take any string value up to the maximum allowed length, whatever its present value may be.

3 For the types Integer, Natural, Boolean, Time, Float, and String see 23.1.1.

8.2.3 Links

Link ::
LINK_TYPE : Link_type_nominator
DESTINATION : [Object_designator]
KEY_ATTRIBUTES : **seq of** Attribute
NON_KEY_ATTRIBUTES : **set of** Attribute
REVERSE : [Link_designator]

Link_designator :: Token

Actual_key = **seq1 of** (Text | Natural)

Link_designators = **set of** Link_designator

Link_selection = Link_type_nominators | VISIBLE_LINK_TYPES | ALL_LINK_TYPES

Link_descriptor = Object_designator * Link_designator

Link_descriptors = **set of** Link_descriptor

Link_set_descriptor = Object_designator * Link_designators

Link_set_descriptors = **set of** Link_set_descriptor

Link_scope = INTERNAL_LINKS | EXTERNAL_LINKS | ALL_LINKS

The key attributes and the non-key attributes are together called the *attributes of the link*. No two attributes of a link have the same attribute type.

Two distinct links of the same type from the same object must have different key attributes (i.e. the two sequences of key attribute values must be different).

The reverse link of the reverse link of a link is that link.

A link is said to be *from* its origin and *to* its destination.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

A *series of links* from object A to object B is a sequence of 1 or more links L1, L2, ..., Ln such that A is the origin of L1, B is the destination of Ln, and otherwise the destination of each link is the origin of the next in sequence.

A link is specified by an object designator which specifies the origin of the link and a link designator which specifies the link relative to the object. For the mapping of link designators to the language bindings, see 23.1.2.4.

NOTES

1 Each link in the object base is a link of exactly one object in the object base; i.e. each link has exactly one origin.

2 When a link is created, so are all its attributes in the global schema. When a new attribute type is applied to the link's type in an SDS, effectively all links of that type gain a new attribute with its initial value. If the application of an attribute type to that link type is removed from all SDSs, the attribute remains on each link of that type until deleted by LINK_DELETE_ATTRIBUTE.

8.3 Types

```
Type = Object_type | Attribute_type | Link_type | Enumeral_type

Type_nominator = Object_type_nominator | Attribute_type_nominator | Link_type_nominator |
Enumeral_type_nominator

Object_type_nominator :: Token

Attribute_type_nominator :: Token

Link_type_nominator :: Token

Enumeral_type_nominator :: Token

Type_nominators = set of Type_nominator

Object_type_nominators = set of Object_type_nominator

Attribute_type_nominators = set of Attribute_type_nominator

Link_type_nominators = set of Link_type_nominator

Type_kind = OBJECT_TYPE | ATTRIBUTE_TYPE | LINK_TYPE | ENUMERAL_TYPE
```

A type is a template defining common basic properties of a set of instances. The *instances* of a type are those whose type nominator identifies that type.

A type is specified by a type nominator, which may be specialized to an object type nominator, an attribute type nominator, a link type nominator, or an enumeral type nominator. A type nominator may be further specialized as follows: if "X" is an object type, attribute type, link type, or enumeral type then 'X_type_nominator' stands for 'Object_type_nominator' etc. with the condition that the value must designate type "X" or a descendant of "X". For the mapping of type nominators to language bindings see 23.1.2.5 and 23.1.2.

8.3.1 Object types

```
Object_type ::
    TYPE_NOMINATOR      : Object_type_nominator
    CONTENTS_TYPE       : [ Contents_type ]
    PARENT_TYPES        : Object_type_nominators
    CHILD_TYPES         : Object_type_nominators
    represented by object_type

Contents_type = FILE_TYPE | PIPE_TYPE | DEVICE_TYPE | AUDIT_FILE_TYPE |
    ACCOUNTING_LOG_TYPE
```


AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

The contents type, if present, specifies the type of contents of instances of the object type. If no contents type is supplied, instances of the object type have no contents.

The parent types define inheritance rules governing the properties of object types in working schema (see 8.5.1). The parent types of an object type, their parent types, and so on, excluding the object type itself, are called the *ancestor types* of the object type.

The child types are the object types which have this object type as parent type. The child types of an object type, their child types, and so on, excluding the object type itself, are called the *descendant types* of the object type.

The parent/child relation between object types forms a directed acyclic graph, with the object type "object" (see 9.1.1) as the root.

8.3.2 Attribute types

```
Attribute_type ::
    TYPE_NOMINATOR              : Attribute_type_nominator
    VALUE_TYPE_IDENTIFIER       : Value_type_identifier
    INITIAL_VALUE               : [ Attribute_value ]
    DUPLICATION                 : Duplication
    represented by attribute_type

Value_type_identifier = INTEGER | NATURAL | BOOLEAN | TIME | FLOAT | STRING |
    Enumeration_value_type_identifier

Enumeration_value_type_identifier = seq1 of Enumeral_type_nominator

Duplication = DUPLICATED | NON_DUPLICATED
```

The value type identifier identifies the *value type* of the instances of the attribute type, i.e. the datatype of their possible attribute values (see table 1). See 23.1.1 for the mapping of values of integers, naturals, Booleans, times, floats, and strings. An enumeration value type identifier is a non-empty sequence of enumeral types.

The initial value, which is a value of the value type, is the initial value of any attribute of this attribute type after creation and before any value has been assigned to it. If no initial value is supplied, the default initial value for the value type is used (see table 1).

If the duplication is DUPLICATED, then every instance of the attribute type is a *duplicable* attribute, i.e. the value of the attribute is copied whenever an object or link with the attribute is copied; if it is NON_DUPLICATED then every instance is a *nonduplicable* attribute, i.e. the value of the copy of the attribute reverts to the initial value.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

Table 1 - Value types

Value type identifier	Value type	Default initial value
INTEGER	Integer	0
NATURAL	Natural	0
BOOLEAN	Boolean	**false**
TIME	Time	1980-01-01T00:00:00Z
FLOAT	Float	0.0
STRING	String	"" (empty string)
Enumeration value type identifier	Enumeral type	1st enumeral type of the enumeration value type identifier

8.3.3 Link types

```
Link_type ::
    TYPE_NOMINATOR                  : Link_type_nominator
    CATEGORY                        : Category
    LOWER_BOUND, UPPER_BOUND        : [ Natural ]
    EXCLUSIVENESS                   : Exclusiveness
    STABILITY                       : Stability
    DUPLICATION                     : Duplication
    KEY_ATTRIBUTE_TYPES             : Key_types
    REVERSE_LINK_TYPE               : [ Link_type_nominator ]
    represented by link_type

Key_types = seq of Attribute_type_nominators

Category = COMPOSITION | EXISTENCE | REFERENCE | DESIGNATION | IMPLICIT

Categories = set of Category

Exclusiveness = SHARABLE | EXCLUSIVE

Stability = ATOMIC_STABLE | COMPOSITE_STABLE | NON_STABLE
```

All instances of a link type have the category, exclusiveness, stability, and duplication of the link type.

The lower bound of a link type defines the number below which the number of links of that link type from any instance of an object type with that link type cannot be reduced. If absent, the lower bound is taken as 0. The lower bound is only checked when an attempt is made to delete a link, so that on creation of an object the number of links of a type may be less than the lower bound for that type.

The upper bound of a link type is an optional natural defining the maximal number of links of that link type from any instance of an object type with that link type. If present, it must be greater than 0 and not less than the lower bound. If absent, there is no upper bound.

A link type is said to be *of cardinality one* if its upper bound is 1. A link type of cardinality one has an empty sequence of key attribute types.

A link type is said to be *of cardinality many* if it is not of cardinality one. A link type of cardinality many has a non-empty sequence of key attribute types.

The sequence of key attribute types defines the attribute types of the sequence of key attributes of an instance of the link type. It does not contain any repeated attribute type nominators. A key attribute has value type Natural or String.

The optional reverse link type is the link type which reverses the link type, i.e. whenever a link of this link type exists from object A to object B, a link of the reverse type exists from object B to object A, and vice versa. The reverse link type is not allowed if the category is DESIGNATION, and must be present otherwise.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

The term *complementary* is used of pairs of links, each having the other's origin as its destination, which are not reverses of each other.

All link types of category IMPLICIT and cardinality many have lower bound 0, no upper bound, and a single key attribute of the predefined attribute type "system_key". The values of "system_key" attributes are implementation-dependent: each such key value is different from the value of every other "system_key" attribute of a link of the same link type from the same object.

All link types of category EXISTENCE and cardinality many have lower bound 0.

The category identifies certain *properties* of instances of the link type, as follows:

- *relevance to the origin*. For a link with this property:
  - The link may be created and deleted explicitly.
  - APPEND_LINKS discretionary access right to the origin is required in order to create the link, and WRITE_LINKS discretionary access right to the origin is required in order to delete the link.
  - The link cannot be created or deleted if its origin is a stable object.
  - The creation and deletion of the link assign the current system time to the last modification time of the origin.
  - The link may have non-key attributes.

  For a link without the relevance to the origin property:
  - The link may only be created and deleted implicitly, i.e. as the reverse of a link with the relevance to the origin property.
  - APPEND_IMPLICIT discretionary access right to the origin is required in order to create the link, and WRITE_IMPLICIT discretionary access right to the origin is required in order to delete the link.
  - The link can be created or deleted even if its origin is a stable object.
  - The creation and deletion of the link have no effect on the last modification time of the origin.
  - The link may not have non-key attributes.
- *referential integrity*. For a link with this property:
  - If the link exists then so does its destination, i.e. the existence of the link prevents the deletion of its destination.
  - The link always has a reverse link with the referential integrity property.
- *existence property*. For a link with this property:
  - An object can be created as destination of the link.
  - The deletion of the link can imply the deletion of its destination.
- *composition property*. For a link with this property:
  - The destination of the link is a component of its origin.

The categories are defined in terms of these properties as follows:

- COMPOSITION: relevance to the origin, referential integrity, existence property, composition property. Links with this category are called *composition links*.
- EXISTENCE: relevance to the origin, referential integrity, existence property. Links with this category are called *existence links*.
- REFERENCE: relevance to the origin, referential integrity. Links with this category are called *reference links*.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

- IMPLICIT: referential integrity. Links with this category are called *implicit links*.
- DESIGNATION: relevance to the origin. Links with this category are called *designation links*.

If the stability of a link type is ATOMIC_STABLE, each instance of the link type is an *atomically stabilizing* link, i.e. the destination of the link (excluding its components other than itself) cannot be modified or deleted.

If the stability of a link type is COMPOSITE_STABLE, each instance of the link type is a *compositely stabilizing* link, i.e. the destination of the link (including its components) cannot be modified or deleted.

If the stability of a link type is NON_STABLE, each instance of the link type is a *nonstabilizing* link, i.e. the existence of the link does not prevent the modification or deletion of its destination or its components.

Modification of an object is defined in 9.1.1. A *stable* object is the destination of an atomically or compositely stabilizing link, or a component of the destination of a compositely stabilizing link.

Exclusiveness applies only to composition link types. If it is EXCLUSIVE, each instance of the link type is an *exclusive* composition link, i.e. no other composition link can share the same destination. If it is SHARABLE, each instance of the link type is a *sharable* composition link, i.e. other composition links can share the same destination.

If duplication is DUPLICATED, each instance is a *duplicable* link, i.e. the link is copied whenever its origin is copied; if it is NON_DUPLICATED, each instance is a *nonduplicable* link, i.e. a copy of the object has no copy of the link. An implicit link cannot be duplicable.

A component of an object is a *duplicable component* if it is the destination of at least one duplicable internal composition link whose origin is either the object or a duplicable component of the object.

A link type of category IMPLICIT or DESIGNATION must be nonstabilizing.

The following relations hold between properties of a link type and of its reverse link type:

- if one link type has category IMPLICIT, then the other does not;
- if one link type has the existence property (i.e. has category EXISTENCE or COMPOSITION) then the other does not;
- if one link type has stability ATOMIC_STABLE or COMPOSITE_STABLE then the other has category IMPLICIT.

A link type of category DESIGNATION cannot have a reverse link type.

Links of the following types are termed *usage designation links*, because they are not checked by the normal security rules: "running_process", "in_working_schema_of" , "consumer_process", "user_identity_of", "adopted_user_group_of", "reserved_by", "locked_by", "lock", "opened_by", "mounted_on", and "listened_to". Usage designation links have the following properties:

- creation or deletion implies only a bitwise write access on the origin object from a mandatory security point of view (see 20.1.8.2);
- creation or deletion requires one unspecified discretionary access permission on the origin object;
- creation or deletion is possible for an object on a read-only volume;
- creation or deletion is possible for a copy object as origin;
- creation or deletion does not require the establishment of locks on the links;
- they are not copied by REPLICATED_OBJECT_DUPLICATE;
- they can be implicitly deleted by network failure and workstation closedown;

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

- creation or deletion does not change the last modification time of the origin object.

Links of the following types are termed *service designation links*, because they indicate that the destination provides a service to the origin (usually a process): "executed_on", "sds_in_working_schema", "consumer_identity", "user_identity", "adopted_user_group", "reserved_message_queue", "open_object", "process_waiting_for", "referenced_object", "adoptable_user_group", "mounted_volume", "is_listener", "notifier", and "executed_static_context". Service designation links have the following properties:

- creation or deletion does not require the establishment of locks on the links;
- they are implicitly deleted by workstation failure;
- for navigation along these links to replicated objects, replication redirection applies to the state of the object base at the time the link was created rather than when it is navigated through.

NOTES

1 The properties of links of various categories are summarized in table 2.

Table 2 - Properties of link categories

Property links	Composition links	Existence links	Reference links	Implicit links	Designation
relevance to origin	yes	yes	yes	no	yes
referential integrity	yes	yes	yes	yes	no
existence	yes	yes	no	no	no
composition	yes	no	no	no	no
atomic stability	optional	optional	optional	no	no
composite stability	optional	optional	optional	no	no
exclusiveness	optional	no	no	no	no
duplication	optional	optional	optional	no	optional
has a reverse link	yes	yes	yes	yes	no

2 The reason why the lower bound of an existence link is 0 is that if there existed an existence link type L with a lower bound of 2, for example, and an object X had two outgoing links of type L, it would be impossible to delete either link directly using LINK_DELETE. Indirect deletion of these links by deletion of object X would also be impossible because X would have outgoing existence links. This means that the destinations of these links could never be deleted. This would be an undesirable situation. The same problem does not exist with composition links because a composite object can be deleted in a single operation, OBJECT-DELETE.

8.3.4 Enumeral types

```
Enumeral_type ::
    TYPE_NOMINATOR     : Enumeral_type_nominator
    represented by enumeral_type
```

An enumeral type is used as a possible value of an enumeration attribute. It has no instances.

8.4 Types in SDS

```
Type_in_sds = Object_type_in_sds | Attribute_type_in_sds | Link_type_in_sds |
    Enumeral_type_in_sds
```

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

```
Type_in_sds_common_part ::
    ASSOCIATED_TYPE   : Type_nominator
    LOCAL_SDS         : Object_designator
    LOCAL_NAME        : [ Name ]

Type_nominator_in_sds = Object_type_nominator_in_sds | Attribute_type_nominator_in_sds |
    Link_type_nominator_in_sds | Enumeral_type_nominator_in_sds

Object_type_nominator_in_sds     :: Token

Attribute_type_nominator_in_sds  :: Token

Link_type_nominator_in_sds       :: Token

Enumeral_type_nominator_in_sds   :: Token

Type_nominators_in_sds = set of Type_nominator_in_sds

Object_type_nominators_in_sds = set of Object_type_nominator_in_sds

Attribute_type_nominators_in_sds = set of Attribute_type_nominator_in_sds

Link_type_nominators_in_sds = set of Link_type_nominator_in_sds

Enumeral_type_nominators_in_sds = set of Enumeral_type_nominator_in_sds

Definition_mode_value = CREATE_MODE | DELETE_MODE | READ_MODE | WRITE_MODE |
NAVIGATE_MODE

Definition_mode_values = set of Definition_mode_value

Definition_modes ::
    USAGE_MODE            : Definition_mode_values
    EXPORT_MODE           : Definition_mode_values
    MAXIMUM_USAGE_MODE    : Definition_mode_values
```

A type in SDS (plural 'types in SDS') is a template defining a set of properties which apply to all instances of its type, in addition to the basic properties of that type. A type in SDS is *associated with* one type; a type is *associated with* one or more types in SDS.

A schema definition set (or SDS) is an object of type "sds" (see 10.1.1), and is specified by an object designator. A type in SDS *belongs to*, or is *in*, a particular SDS, called its local SDS.

The local name identifies the type in SDS, and hence the associated type, uniquely within the local SDS. The *complete name* of a type in SDS is the name of the SDS, followed by a hyphen '-', followed by the local name of the type in SDS.

The definition modes specify restrictions on the usage of the type in SDS. The usage mode specifies the permitted kinds of access to instances of the type in SDS by a process which has adopted its local SDS in its working schema. The export mode specifies the maximum usage mode of the copy of the type in SDS which is created when the type in SDS is exported to another SDS; it is a subset of the usage mode. The maximum usage mode specifies which definition mode values can be included in the usage mode and export mode; it is set on creation of the type in SDS and cannot be changed. The definition modes of a link and of its reverse must be the same. Enumeral types in SDS do not have definition modes.

The accesses controlled by definition modes are as follows.

- READ_MODE controls reading from attributes by the operations OBJECT_GET_ATTRIBUTE, OBJECT_GET_SEVERAL_ATTRIBUTES, LINK_GET_ATTRIBUTE, and LINK_GET_SEVERAL_ATTRIBUTES.
- WRITE_MODE controls writing to attributes by the operations OBJECT_SET_ATTRIBUTE, OBJECT_SET_SEVERAL_ATTRIBUTES, OBJECT_RESET_ATTRIBUTE, LINK_SET_ATTRIBUTE, LINK_SET_SEVERAL_ATTRIBUTES, and LINK_RESET_ATTRIBUTE.

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

- CREATE_MODE controls creation of objects and links by the operations OBJECT_CREATE, OBJECT_COPY, OBJECT_CONVERT, VERSION_REVISE, VERSION_SNAPSHOT, DEVICE_CREATE, and PROCESS_CREATE; and creation of links by the operations LINK_CREATE and LINK_REPLACE.
- DELETE_MODE controls deletion of objects and links by the operation OBJECT_DELETE and deletion of links by the operations LINK_DELETE and LINK_REPLACE.
- NAVIGATE_MODE controls the use of link references in pathnames in the evaluation of object references (see 23.1.2.2).

Types in SDS are specialized to object types in SDS, attribute types in SDS, link types in SDS, and enumeral types in SDS; the associated types are object types, attribute types, link types, and enumeral types respectively.

A type in SDS is specified by a type nominator in SDS, which may be specialized to an object type nominator in SDS, an attribute type nominator in SDS, a link type nominator in SDS, or an enumeral type nominator in SDS. A type nominator in SDS may be further specialized as follows: if "X" is an object type, attribute type, link type, or enumeral type then 'X_type_nominator_in_sds' stands for 'Object_type_nominator_in_sds' etc. with the condition that the value must designate a type in SDS associated with type "X" or a descendant of "X". For the mapping of type nominators in SDS to language bindings see 23.1.2.5.

NOTE - The properties of a type and of an associated type in SDS can be specified by means of the Data Definition Language (see annex B).

8.4.1 Object types in SDS

```
Object_type_in_sds :: Type_in_sds_common_part &&
    DIRECT_ATTRIBUTE_TYPES_IN_SDS      : Attribute_type_nominators_in_sds
    DIRECT_OUTGOING_LINK_TYPES_IN_SDS  : Link_type_nominators_in_sds
    DIRECT_COMPONENT_TYPES_IN_SDS      : Object_type_nominators_in_sds
    DEFINITION_MODES                   : Definition_modes
    represented by object_type_in_sds
```

The only allowed definition mode value for an object type in SDS is CREATE_MODE.

The direct attribute types in SDS must be in the same SDS as the object type in SDS. They participate in the definition of the visible attribute types of object types in working schema; see 8.5.1.

The direct outgoing link types in SDS must be in the same SDS as the object type in SDS. They participate in the definition of the visible link types of object types in working schema; see 8.5.1. The object type in SDS is called the *origin object type in SDS* of each of the direct outgoing link types in SDS.

The direct component types in SDS must be in the same SDS as the object type in SDS. They participate in the definition of the direct component types of object types in working schema; see 8.5.1.

8.4.2 Attribute types in SDS

```
Attribute_type_in_sds :: Type_in_sds_common_part &&
    DEFINITION_MODES   : Definition_modes
    represented by attribute_type_in_sds
```

The only allowed definition mode values for an attribute type in SDS are READ_MODE and WRITE_MODE.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

8.4.3 Link types in SDS

```
Link_type_in_sds :: Type_in_sds_common_part &&
    DESTINATION_OBJECT_TYPES_IN_SDS : Object_type_nominators_in_sds
    NON_KEY_ATTRIBUTE_TYPES_IN_SDS  : Attribute_type_nominators_in_sds
    DEFINITION_MODES                : Definition_modes
    represented by link_type_in_sds
```

The only allowed definition mode values for a link type in SDS are CREATE_MODE, DELETE_MODE, and NAVIGATE_MODE.

The destination object types in SDS must be in the same SDS as the link type in SDS. They participate in the definition of the destination object types of link types in working schema; see 8.5.3.

The non-key attribute types in SDS must be in the same SDS as the link type in SDS. They participate in the definition of the visible attribute types of link types in working schema; see 8.5.3.

8.4.4 Enumeral types in SDS

```
Enumeral_type_in_sds :: Type_in_sds_common_part &&
    IMAGE : Text
    represented by enumeral_type_in_sds
```

An enumeral type in SDS associates with the enumeral type a string called its image.

8.5 Types in working schema

```
Type_in_working_schema = Object_type_in_working_schema |
    Attribute_type_in_working_schema | Link_type_in_working_schema |
    Enumeral_type_in_working_schema

Type_in_working_schema_common_part ::
    ASSOCIATED_TYPE          : Type_nominator
    CONSTITUENT_TYPES_IN_SDS : seq of (Composite_name * Type_nominator_in_sds)

Composite_name ::
    SDS_NAME   : Name
    LOCAL_NAME : [ Name ]
```

A type in working schema is a template defining common properties for a set of instances of its type. The properties of a type in working schema are derived from the properties of one or more types in SDS (see 8.5.1 to 8.5.4). Types in working schema occur in working schemas, see 8.1. For the construction of working schemas, see 13.2.12.

The constituent types in SDS of a type in working schema must all have the same associated type, which is the type *associated with* the type in working schema.

A type in working schema has several composite names, one for each constituent type in SDS. For each composite name, the SDS name is the name of the local SDS of the corresponding type in SDS, and the local name, if any, is the local name of the type in SDS in its local SDS.

Let C1 and C2 be composite names, and T1 and T2 be type nominators in SDS. Then for any two constituent types in SDS (C1, T1), (C2, T2) of a type in working schema, if the SDS name of C1 precedes the SDS name of C2 in the SDS names of the working schema containing the type in working schema, then (C1, T1) precedes (C2, T2).

Types in working schema are specialized to object types in working schema, attribute types in working schema, link types in working schema, and enumeral types in working schema; their associated types are object types, attribute types, link types, and enumeral types respectively.

The value of a type in SDS cannot be changed while it is part of a type in working schema.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

A type in working schema is specified by a type nominator, see 8.3.

8.5.1 Object types in working schema

```
Object_type_in_working_schema :: Type_in_working_schema_common_part &&
    CHILD_TYPES                 : Object_type_nominators
    PARENT_TYPES                : Object_type_nominators
    APPLIED_ATTRIBUTE_TYPES     : Attribute_type_nominators
    APPLIED_LINK_TYPES          : Link_type_nominators
    VISIBLE_ATTRIBUTE_TYPES     : Attribute_type_nominators
    VISIBLE_LINK_TYPES          : Link_type_nominators
    DIRECT_COMPONENT_TYPES      : Object_type_nominators
    USAGE_MODES                 : Definition_mode_values
```

The set of constituent types in SDS of the child types is the union of the sets of child types of the constituent types in SDS of the type in working schema.

The set of constituent types in SDS of the parent types is the union of the sets of parent types of constituent types in SDS of the type in working schema.

The applied attribute types are the attribute types in working schema which have a constituent type in SDS of a direct attribute type in SDS of one of the constituent types in SDS of the object type in working schema.

The applied link types are the link types in working schema which have a constituent type in SDS of a direct outgoing link type in SDS of one of the constituent types in SDS of the object type in working schema.

The direct component types are the object types in working schema which have a constituent type in SDS of a direct component type in SDS of one of the constituent types in SDS of the object type in working schema.

The set of visible attribute types is the union of the set of applied attribute types and the sets of the visible attribute types of all the parent types.

The set of visible link types is the union of the set of applied link types and the sets of the visible link types of all the parent types.

The constituent types in SDS must be object types in SDS.

If the type of an object is not visible, the object is considered as an instance of any of its object type's ancestor types which is visible.

The set of usage modes is the union of the sets of definition modes of all constituent types in SDS of the object type in working schema.

8.5.2 Attribute types in working schema

```
Attribute_type_in_working_schema :: Type_in_working_schema_common_part &&
    USAGE_MODES    : set of Definition_mode_values
```

The constituent types in SDS must be attribute types in SDS.

The set of usage modes is the union of the sets of definition modes of all constituent types in SDS of the attribute type in working schema.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

8.5.3 Link types in working schema

```
Link_type_in_working_schema :: Type_in_working_schema_common_part &&
    DESTINATION_OBJECT_TYPES          : Object_type_nominators
    VISIBLE_DESTINATION_OBJECT_TYPES  : Object_type_nominators
    KEY_ATTRIBUTE_TYPES               : Key_types
    APPLIED_ATTRIBUTE_TYPES           : Attribute_type_nominators
    REVERSE                           : [ Link_type_nominator ]
    USAGE_MODES                       : Definition_mode_values
```

The set of constituent types in SDS of the applied attribute types is the union of the sets of non-key attribute types of the constituent types in SDS of the link type in working schema.

The set of constituent types in SDS of the destination object types is the union of the sets of destination object types of the constituent types in SDS of the link type in working schema.

The constituent types in SDS must be link types in SDS.

The set of visible destination object types is the union of the set of destination object types and the set of visible descendants of the visible destination object types.

The sequence of key attribute types is the same as the sequence of key attribute types of the associated type.

The set of usage modes is the union of the sets of definition modes of all constituent types in SDS of the link type in working schema.

8.5.4 Enumeral types in working schema

```
Enumeral_type_in_working_schema ::
    Type_in_working_schema_common_part &&
    IMAGE : Text
```

The image of an enumeral type in working schema T1 is the image of the first of its types in SDS which has an image, unless another enumeral type in working schema T2 belonging to the same enumeration attribute type in working schema as T1 but with a lower position already has the same image, in which case T1 has no image.

The constituent types in SDS must be enumeral types in SDS.

8.6 Types in global schema

The *global schema* is the working schema constituted by all the SDSs of a PCTE installation; the order is irrelevant as it affects only the type names, which are of no concern here. A *type in global schema* is a type in working schema in the global schema; it follows that each type is associated with one type in global schema. The global schema is a notional working schema used to state the following consistency rules applying to the whole object base; it is not necessarily the working schema of any process.

An object must be compatible with its associated object type in global schema, i.e.:

- The link types in global schema of the links of the object must be among the visible link types in global schema of the object type in global schema.
- The attribute types in global schema of the attributes of the object must be among the visible attribute types in global schema of the object type in global schema.
- The object types in global schema of the direct components of the object must be among the direct component object types in global schema of the object type in global schema.
- The preferred link type of the object, if present, must be one of the applied link types of the object type in global schema.

AVA Documents Document Sharing Hub
Full document details: ISO_IEC 13719-1_1995

- The preferred link key of the object, if present, must have the same value types (String or Natural), in the same order, as the key attribute types of the preferred link type of the object.

A link must be compatible with its associated link type in global schema, i.e.:

- The object type in global schema of the destination of the link must be among the visible destination types of the link type in global schema.
- The key attributes of the link, if any, must have the same value types (String or Natural) in the same order as the key attribute types of the link type in global schema.
- The non-key attributes of the link must be among the applied attribute types of the link type in global schema.
- The link type in global schema of the reverse link, if any, must be the reverse of the link type in global schema.

8.7 Operations

8.7.1 Calling process

The operations defined in clauses 9 to 22 take effect when they are *executed* by a process (see 13.1.4). The process is known as the *calling process* of the operation. The effects on the state of the PCTE installation are global, i.e. can be observed by other processes. Results returned by operations which are designators are local to the calling process.

8.7.2 Direct and indirect effects

The effects of an operation on the state of the PCTE installation comprise direct effects and indirect effects. *Direct effects* are described in the relevant operation descriptions (including the error descriptions). *Indirect effects* are described elsewhere in clauses 9 to 22. The operations of clause 23 do not affect the state.

Indirect effects occur by means of *events*. Events are of several classes, described below. An operation may *raise* an event. Depending on the state of the PCTE installation, the raising of an event may result either in the effect of another operation being different to what it would otherwise be, or in some other change of state.

An operation has a finite non-zero duration, and an event that is raised during the operation may have an effect on that operation.

In general, the raising of an event is not explicitly described in the operation that raises the event, but instead in the part of the interface definition that may be affected by the event. It must nevertheless be understood that the description of an operation may need to be implicitly extended by the description of the raising of events. The processing of an event takes place asynchronously.

There are several different classes of event, as described below.

- *Access event.* This is described in clause 15. Access events are raised by operations which perform certain kinds of access to objects. If an appropriate notifier has been established, then the raising of the event causes an appropriate message to be sent.
- *Lock release event.* A lock release event occurs when a lock is released (see 16.1.8). If some other operation is waiting to acquire a lock on the same resource, then that operation may proceed. If there is more than one such operation then which operation acquires the lock is implementation-dependent. There is no further description of this event in this part of ISO/IEC 13719.
- *Process timeout event.* This event is raised when the duration of an operation has exceeded the process timeout value for that process. When this event is raised, the error condition

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

OPERATION_HAS_TIMED_OUT holds for that operation, and the operation terminates with that error. This event is described in 13.1.4.

- *Process alarm event.* This event is raised when the time left until alarm has expired. When this event is raised, a message of message type WAKE is sent to the process and the process is resumed. This event and its effect are described in 13.1.4 and 13.2.6.
- *Interrupt operation event.* This is described in 13.2.4. This event is raised by PROCESS_INTERRUPT_OPERATION or by PROCESS_TERMINATE. If the interrupted process is executing an operation or waiting for an event when this error is raised, then the error condition OPERATION_IS_INTERRUPTED holds for that operation and it terminates with that error.
- *Audit event.* These events are described in clause 21. Audit events are raised by operations which carry out object access, and for exploiting audit records, copying audit records, carrying out certain security operations, and at explicit user request (see 21.2.5). If the event type is selected and auditing is enabled (or the event type is always audited) then an audit record is written as described in 21.1.1.
- *Accounting event.* Accounting events are divided into start events and end events. These are raised as a result of certain operations, and if the resource is accountable and accounting is enabled then an accounting record is written (see 22.1.2).
- *Message queue event.* These events are described in 14.1. They are raised by the appearance in a message queue of a message, which may be caused by MESSAGE_SEND_WAIT, MESSAGE_SEND_NO_WAIT, or QUEUE_RESTORE. If there is a handler for the event then that handler is executed.
- *Resource availability event.* These events do not occur as a result of operations, but may occur at any moment. They model changes in the availability of hardware resources. The effect of a resource availability event on the directly affected objects is implementation-dependent. The set of directly affected objects for an event is implementation-defined. The effect on access from other objects is defined as follows for the various resource availability events.
  - *Volume failure*: an accessible volume becomes inaccessible. Attempted access to objects on the volume (including replicas in the case of an administration volume) fails with the error OBJECT_IS_INACCESSIBLE. Attempts to commit transactions which have started and which involve objects on the volume also fail.
  - *Device failure*: an accessible device becomes inaccessible. Attempted access to the file contents of the device fails. Volume failure is raised for any volume mounted on the device.
  - *Network failure*: an accessible workstation becomes inaccessible. There is no distinction between a workstation failing and a network failing so that communication with the workstation is lost. The inaccessible workstation ceases to be in its current network partition. Associated devices and volumes become inaccessible with consequences as above.
  - *Network repair*: a workstation joins a network partition. This has no immediate effect, but the workstation becomes accessible when the other conditions are met.
- *Process termination event.* This event is raised when a process is terminated by PROCESS_TERMINATE, explicitly or implicitly or by normal or abnormal process termination. If a PROCESS_WAIT_FOR_CHILD or PROCESS_WAIT_FOR_ANY_CHILD operation of the parent process is waiting for that process or any sibling process to terminate, then it may proceed. If more than one such operation exists (for different threads) then all may proceed.
- *Data available event.* This event is raised when data is written to a device contents, pipe contents, or message queue. If an operation is waiting on a CONTENTS_READ on a pipe or device which is not non-blocking, and data is written to that pipe or device, then that operation may proceed. If an operation is waiting on a MESSAGE_RECEIVE_WAIT on a message queue, and a message is received of a type included in the set of types specified by that

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

MESSAGE_RECEIVE_WAIT, then the operation may proceed. If more than one operation is waiting on that event, which operation receives the data and ceases to wait is implementation-dependent. AUDIT_FILE_READ and ACCOUNTING_LOG_READ do not wait for data to be available.

- *Data space available event.* This event is raised when space becomes available in a device contents, pipe contents, or message queue. If an operation is waiting on a CONTENTS_WRITE on a pipe or device which is not non-blocking, and data is removed from that pipe or device, then that operation may proceed if sufficient space has been made available. If an operation is waiting on a MESSAGE_SEND_WAIT on a message queue and a message is removed from the queue then the operation may proceed if sufficient space has been made available. If an operation is waiting to write to the audit file or accounting log and the audit file or accounting log becomes available, then the operation proceeds. If more than one operation is waiting on that event for a contents or message queue, which operation writes or sends the data and ceases to wait is implementation-dependent.
- *Security attribute change.* This event is raised by OBJECT_SET_CONFIDENTIALITY_LABEL, OBJECT_SET_INTEGRITY_LABEL, or OBJECT_SET_ACL_ENTRY, or changes to labels as a result of floating. If a CONTENTS_READ, CONTENTS_WRITE, MESSAGE_RECEIVE_WAIT or MESSAGE_SEND_WAIT operation is waiting and a security attribute changes such that the process no longer has permission to access the contents or message queue object in the required mode, then the operation ceases to wait and terminates in an appropriate mandatory or discretionary access mode error (see C.4).

8.7.3 Errors

Execution of an operation may *terminate* after carrying out the *normal behaviour* as described in the main subclause, or may terminate in an *error*.

The list of errors in the subclause **Errors** of each abstract operation definition defines the set of *error conditions* which apply to that operation. Other error conditions, which apply to several operations, are defined in clause 23. The error conditions OPERATION_HAS_TIMED_OUT and OPERATION_IS_INTERRUPTED (see 8.7.2) and SECURITY_POLICY_WOULD_BE_VIOLATED (see 20.1.8) apply to all operations. If an operation terminates in an error then the associated error condition holds. If any error condition holds then the operation terminates; if none of the error conditions hold, the normal behaviour occurs.

Error conditions are distinguished for the purpose of helping tool writers. Language bindings may add further error conditions. An implementation may not add further error conditions, except as specified in this part of ISO/IEC 13719.

No precedences are defined in this part of ISO/IEC 13719 between error conditions which hold simultaneously. Implementations which aim for high security must define such a precedence so as to address the problem of covert channels.

Error conditions arising from type mismatches between actual and formal parameters of operations are not explicitly defined in the abstract operation definitions. Language bindings may need to make these error conditions explicit, depending on the strength of type-checking provided by the language. This does not apply to the following cases:

- a check by an operation that an object belongs to a particular subset of instances of a type, e.g. that a security group is not a subgroup, or that an attribute is not applied to a specified object;
- a check by an operation where a specialization of Object_designator is specified and an object reference is supplied (see 23.1.2.2).

If an operation terminates with an error condition, then the operation may have acquired some locks. The locks acquired are implementation-dependent, but in no case may a lock be acquired on a resource (object or link) which is stronger than the lock that would be acquired on that

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

resource if the normal behaviour had occurred. Their duration is determined in the same way as for other locks. No other state changes occur, except that possibly auditing and accounting records are created.

8.7.4 Operation serializability

In general, operations are serializable with all other concurrent operations. An operation may be considered to be composed of one or more atomic *actions* which change the state of the PCTE installation. That a set of operations is serializable means that for each operation a single point in time can be determined, lying between the actual time the operation is called and the time of the return from the operation, where all the actions of the operation can be deemed to take place and without any different effect on the state of the PCTE installation to that which actually occurs. This point of time is called the *nominal serialization point*. The following specific exceptions to serializability apply.

- If an operation enters a waiting state then the actions before the operation waits constitute one operation for purposes of serialization and the actions after leaving the waiting state until entering a further waiting state or the end of the operation constitute another operation.
- The values of the "last_access_time", "last_modification_time", "last_change_time", "last_composite_access_time", "last_composite_modification_time", and "last_composite_change_time" attributes are updated within an operation to a point of time between the start and end of the operation and not necessarily to any nominal serialization point. The time values set on different objects by a single operation are not necessarily the same.
- If PROCESS_INTERRUPT_OPERATION is used on a process between the start of an operation and the nominal serialization point then the operation is interrupted; if it is used after the nominal serialization point then it has no effect.
- Serializability does not apply to PROCESS_SUSPEND for the calling process; nor to WORKSTATION_CHANGE_CONNECTION with the parameter *force* set to **true** and any affected concurrent operations.
- PROCESS_TERMINATE interrupts other ongoing operations (if any) in the same way as PROCESS_INTERRUPT_OPERATION.

NOTES

1 Serializability is often enforced by locking. However, this is not true for two or more operations running on behalf of the same activity or on behalf of competing unprotected activities. As an example of operation serializability, consider two concurrent invocations of OBJECT_MOVE on the same object, moving it to two different volumes. The result should be that the entire object resides on one or the other volume, rather than some components residing on each volume according to the order in which they were moved.

2 Evaluation of parameters which are references counts as part of operation execution for serialization.

9 Object management

9.1 Object management concepts

9.1.1 The basic type "object"

```
sds system:
volume_identifier: (read) non_duplicated natural;
locked_link_name: (read) string;
exact_identifier: (read) non_duplicated string;
number: natural;
name: string;
```


AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

```
    system_key: (read) natural;

    object: with
    attribute
        exact_identifier;
        volume_identifier;
        replicated_state: (read) non_duplicated enumeration (NORMAL, MASTER, COPY) :=
            NORMAL;
        last_access_time: (read) non_duplicated time;
        last_modification_time: (read) non_duplicated time;
        last_change_time: (read) non_duplicated time;
        last_composite_access_time: (read) non_duplicated time;
        last_composite_modification_time: (read) non_duplicated time;
        last_composite_change_time: (read) non_duplicated time;
        num_incoming_links: (read) non_duplicated integer;
        num_incoming_composition_links: (read) non_duplicated natural;
        num_incoming_existence_links: (read) non_duplicated natural;
        num_incoming_reference_links: (read) non_duplicated natural;
        num_incoming_stabilizing_links: (read) non_duplicated natural;
        num_outgoing_composition_links: (read) non_duplicated natural;
        num_outgoing_existence_links: (read) non_duplicated natural;
    link
        predecessor: (navigate) non_duplicated composite stable existence link
            (predecessor_number: natural) to object reverse successor;
        successor: (navigate) implicit link (system_key) to object reverse predecessor;
        opened_by: (navigate) non_duplicated designation link (number) to process;
        locked_by: (navigate) non_duplicated designation link (number ) to activity with
        attribute
            locked_link_name;
        end locked_by;
    end object;

    end system;
```

"Object" is the common ancestor type of all objects in the object base.

The exact identifier uniquely identifies the object in the object bases of all PCTE installations. It is composed of a prefix and a suffix separated by ':' (colon). The prefix is the same for all objects created within a PCTE installation. The suffix uniquely identifies the object within the object base of a particular PCTE installation. The exact identifier of an object that has been deleted is never reassigned to an object created later.

The volume identifier identifies the volume on which the object resides, or, for a copy object, on which it is a replica. It uniquely identifies a volume within a PCTE installation.

The replicated state indicates whether the object is normal, a master or a copy (see 17.1). It is MASTER for the master of a replicated object, COPY for a copy of a replicated object, and NORMAL for a non-replicated object. This attribute can be changed only by the operations which manage replicated objects.

The last access time is the date and time of day of the last read access to the contents of the object. It is set to the system time when the object is created and by the following operations (unless the object is on a read-only volume or is a component of an object on a read-only volume):

- QUEUE_RESTORE (*queue*, *file*) for *file*;
- CONTENTS_READ;
- AUDIT_FILE_READ;
- ACCOUNTING_RECORD_READ;
- CONTENTS_COPY_TO_FOREIGN_SYSTEM (*file*, *foreign_system*, *foreign_parameters*, *foreign_name*) for *file*.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

The last modification time is the date and time of day of the last *modification* to the object. It is set to the system time when the object is created and by the following operations:

- LINK_CREATE and LINK_REPLACE for the origin of the created link when the created link is not implicit;
- LINK_DELETE and LINK_REPLACE for the origin of the deleted link when the deleted link is not implicit;
- any operation which results in the creation or deletion of a link which is not implicit, except for usage designation links and "object_on_volume" links;
- OBJECT_CONVERT;
- OBJECT_SET_ATTRIBUTE and OBJECT_SET_SEVERAL_ATTRIBUTES;
- LINK_SET_ATTRIBUTE and LINK_SET_SEVERAL_ATTRIBUTES, for the origins of the links;
- OBJECT_SET_PREFERENCE;
- QUEUE_SAVE (*queue*, *file*) for *file*;
- CONTENTS_WRITE and CONTENTS_TRUNCATE;
- CONTENTS_COPY_FROM_FOREIGN_SYSTEM (*file*, *foreign_system*, *foreign_parameters*, *foreign_name*) for *file*;
- any operation resulting in the creation of an audit record for the audit file;
- any operation resulting in the creation of an accounting record for the accounting log;

The last change time is the date and time of day of the last *change* to the object. It is set by any operation which sets the last modification time, and the following operations:

- creation of an implicit link;
- deletion of an implicit link;
- OBJECT_MOVE for an object which has been moved to another volume;
- operations which change the discretionary access control lists of an object;
- operations which change the mandatory labels of an object;
- operations which change the mandatory label ranges of multi-level secure devices (see 20.1.5);
- PROCESS_SET_CONFIDENTIALITY_LABEL (*process*, *label*) for *process*;
- PROCESS_SET_INTEGRITY_LABEL (*process*, *label*) for *process*.

The last composite access time is the date and time of day of the last read access to the contents of the object or of any component of the object (but is not updated if the object or component is on a read-only volume).

The last composite modification time is the date and time of day of the last modification to the object or to any component of the object.

The last composite change time is the date and time of day of the last change made to the object or to any component of the object.

An operation which updates the last modification time of an object is said to *atomically modify* the object. An operation which updates the last composite modification time of an object is said to *compositely modify* the object.

The num (number of) incoming links is the number of non-designation links to the object (and is also the number of non-designation links of the object since every non-designation link has a reverse link).

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

The num (number of) incoming composition links is the number of composition links to the object.

The num (number of) incoming existence links is the number of existence links to the object.

The num (number of) incoming reference links is the number of reference links to the object.

The num (number of) incoming stabilizing links is the number of atomically stabilizing links to the object plus the number of compositely stabilizing links to the object and to its outer objects.

The num (number of) outgoing composition links is the number of composition links of the object.

The num (number of) outgoing existence links is the number of existence links of the object.

The destinations of the "predecessor" links are the immediate predecessor versions of the object; the destinations of the "successor" links are the immediate successor versions of the object. These are used in version control operations; see 9.4. The directed graph of versions created by these links must be acyclic.

The destination of the "opened_by" link is the process that opened the object; see 12.1.

The destination of the "locked_by" link is the activity that has locked the object or a link of the object; see 16.1.2.

There are also attributes defined in the security SDS representing the security properties of the object (see 19.1.2 and 20.1.1); and attributes defined in the accounting SDS representing the accounting properties of the object (see 22.1.1).

The attribute types "number", "name" and "system_key" are predefined. "number" and "name" are used for numeric and string keys, respectively; names are non-empty. "system_key" is the attribute type of system-assigned keys of implicit links (see 8.3.3).

NOTES

1 The prefix of the object exact identifier is intended to be unique among all PCTE installations, past, present, and future; but the administration of prefix assignment is outside the scope of this part of ISO/IEC 13719.

2 The last composite access, modification, and change time may be calculated when required as the most recent of the last access, modification, and change time respectively of the object and its components.

9.1.2 The common root

```
sds system:

common_root: child type of object with
link
    archives: (navigate) existence link to archive_directory reverse archives_of;
    execution_sites: (navigate) existence link to execution_site_directory reverse
        execution_sites_of;
    replica_sets: (navigate) existence link to replica_set_directory reverse replica_sets_of;
    volumes: (navigate) existence link to volume_directory reverse volumes_of;
end common_root;

end system;
```

The common root has an existence link to each of the *administrative objects* of the PCTE installation: the SDS directory (see 10.1.1), the volume directory (see 11.1.1), the archive directory (see 11.1.4), the replica set directory (see 17.1.1), the execution site directory, (see 18.1.1), the security group directory (see 19.1.1), the mandatory directory (see 20.1.1), and the accounting directory (see 22.1.1).

The common root and the administrative objects are predefined replicated objects (see 17.1.4).

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

9.1.3 Datatypes for object management

Type_ancestry = EQUAL_TYPE | ANCESTOR_TYPE | DESCENDANT_TYPE | UNRELATED_TYPE

Version_relation = ANCESTOR_VSN | DESCENDANT_VSN | SAME_VSN | RELATED_VSN | UNRELATED_VSN

These datatypes are used as parameter and result types of operations in 9.3 and 9.4.

9.2 Link operations

9.2.1 LINK_CREATE

```
LINK_CREATE (
    origin          : Object_designator,
    new_link        : Link_designator,
    dest            : Object_designator,
    reverse_key     : [ Actual_key ]
)
```

LINK_CREATE creates a new link *link* of *origin* as follows:

- the link type is as specified by the link designator *new_link*;
- the destination is the object *dest*;
- the key attributes are as specified by the link designator *new_link*;
- the non-key attributes are set to their initial values;
- the category, exclusiveness, stability, and duplication are set according to the link type.

If the type of *link* has a reverse link type, LINK_CREATE also creates the reverse link *reverse_link* of *link* and adds it to the links of *dest*:

- the link type of *reverse_link* is the reverse of the link type of *link*;
- the destination of *reverse_link* is *origin*;
- the category, exclusiveness, stability, and duplication of *reverse_link* are set according to the link type.
- the non-key attributes of *reverse_link* are set to their initial values;
- if the type of *reverse_link* is of cardinality many and of category IMPLICIT then the key attribute of *reverse_link* is set to a new system-generated key.

If the type of *reverse_link* is of category COMPOSITION, REFERENCE or EXISTENCE, two cases arise:

- if *dest* has a preferred link type which is the link type of the reverse link, then the key attributes of *reverse_link* are derived from *reverse_key* and from the preferred link key of *dest*, if any, as defined in 23.1.2.7;
- if *dest* has no preferred link type or if the preferred link type of *dest* is not the link type of the reverse link, then the key of the reverse link is set to *reverse_key*.

If *new_link* is a composition link, then any security group that has OWNER granted or denied to *origin* has OWNER granted or denied respectively to *dest*; similarly if *reverse_link* is a composition link, then any security group that has OWNER granted or denied to *dest* has OWNER granted or denied respectively to *origin*. This requires the process to have OWNER rights on *dest* or *origin* respectively. See 19.1.2 for details.

Write locks of the default mode are obtained on the new links. A read lock of the default mode is obtained on *origin* if the interpretation of *new_link* implies the evaluation of any '+' or '++' key

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

attribute values (see 23.1.2.7). A read lock of the default mode is obtained on *dest* if the interpretation of *reverse_key* implies the evaluation of any '+' or '++' key attribute values.

A write lock of the default mode is obtained on *dest* and each of its components if the OWNER discretionary access right is granted or denied for one or more groups to *origin*, and different OWNER discretionary access rights exist for one or more of those same groups to *dest*.

Errors

ACCESS_ERRORS (*origin*, ATOMIC, MODIFY, APPEND_LINKS)

If *reverse_link* is implicit:
ACCESS_ERRORS (*dest*, ATOMIC, CHANGE, APPEND_IMPLICIT)

If *reverse_link* is not implicit:
ACCESS_ERRORS (*dest*, ATOMIC, MODIFY, APPEND_LINKS)

If *new_link* is atomically stabilizing:
ACCESS_ERRORS (*dest*, ATOMIC, CHANGE, STABILIZE)

If *new_link* is compositely stabilizing:
ACCESS_ERRORS (*dest*, COMPOSITE, CHANGE, STABILIZE)

CATEGORY_IS_BAD (*origin*, *new_link*, (COMPOSITION, EXISTENCE , REFERENCE, DESIGNATION))

COMPONENT_ADDITION_ERRORS (*dest*, *new_link*)

COMPONENT_ADDITION_ERRORS (*origin*, *reverse_link*)

DESTINATION_OBJECT_TYPE_IS_INVALID (*origin*, *new_link*, *dest*)

LINK_EXISTS (*origin*, *new_link*)

If *link* is atomically or compositely stabilizing:
OBJECT_CANNOT_BE_STABILIZED (*dest*)

If *link* is compositely stabilizing:
OBJECT_CANNOT_BE_STABILIZED (component of *dest*)

REVERSE_KEY_IS_BAD (*origin*, *new_link*, *dest*, *reverse_key*)

REVERSE_KEY_IS_NOT_SUPPLIED (*origin*, *new_link*, *dest*)

REVERSE_KEY_IS_SUPPLIED (*reverse_key*)

REVERSE_LINK_EXISTS (*origin*, *new_link*, *dest*, *reverse_key*)

UPPER_BOUND_WOULD_BE_VIOLATED (*dest*, *reverse_link*)

UPPER_BOUND_WOULD_BE_VIOLATED (*origin*, *new_link*)

USAGE_MODE_ON_LINK_TYPE_WOULD_BE_VIOLATED (*origin*, *new_link*, CREATE_MODE)

9.2.2 LINK_DELETE

```
LINK_DELETE (
    origin   : Object_designator,
    link     : Link_designator
)
```

LINK_DELETE deletes the link specified by *origin* and *link*.

Let *dest* be the destination of *link*, and *reverse_link* be the reverse link of *link* (if any). LINK_DELETE deletes *link* from the links of *origin* and deletes *reverse_link* (if any) from the links of *dest*.

dest is deleted from the object base if *link* is the last composition or existence link to *dest*.

origin is deleted from the object base if *reverse_link* is the last composition or existence link to *origin*.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

For each deleted object the "object_on_volume" link from the volume on which the deleted object was residing to the deleted object is also deleted.

If either deleted object is opened by one or more processes (see 12.1), the deletion of its contents is postponed until all processes have closed the contents. An operation using a contents handle to access its contents is not affected by the deletion until the contents handle is closed.

Write locks of the default mode are obtained on the deleted objects (if any) and on the deleted links except the "object_on_volume" link. A read lock of the default mode is obtained on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*origin*, ATOMIC, MODIFY, WRITE_LINKS)

If *link* is compositely stabilizing:
ACCESS_ERRORS (*dest*, COMPOSITE, CHANGE, STABILIZE)

If *reverse_link* is implicit:
ACCESS_ERRORS (*dest*, ATOMIC, CHANGE, WRITE_IMPLICIT)

If *reverse_link* is not implicit:
ACCESS_ERRORS (*dest*, ATOMIC, MODIFY, WRITE_LINKS)

If *link* is the last composition or existence link to *dest*:
ACCESS_ERRORS (*dest*, ATOMIC, MODIFY, DELETE)

If *reverse_link* is the last composition or existence link to *origin*:
ACCESS_ERRORS (*origin*, ATOMIC, MODIFY, DELETE)

For each origin X of an implicit link to a deleted object:
ACCESS_ERRORS (X, ATOMIC, CHANGE, WRITE_IMPLICIT)

For each compositely stabilizing link L of a deleted object:
ACCESS_ERRORS (destination of L, COMPOSITE, CHANGE)

CATEGORY_IS_BAD (*origin*, *link*, (COMPOSITION, EXISTENCE, REFERENCE, DESIGNATION))

If *link* is not a designation link:
DESTINATION_OBJECT_TYPE_IS_INVALID (*origin*, *link*, *dest*)

LOWER_BOUND_WOULD_BE_VIOLATED (*origin*, *link*)

LOWER_BOUND_WOULD_BE_VIOLATED (*dest*, *reverse_link*)

If *reverse_link* is the last existence or composition link to *origin*:
OBJECT_HAS_LINKS_PREVENTING_DELETION (*origin*)

If *link* is the last existence or composition link to *dest*:
OBJECT_HAS_LINKS_PREVENTING_DELETION (*dest*)

If *link* is the last composition or existence link to *dest*:
OBJECT_IS_IN_USE_FOR_DELETE (*dest*)

If *reverse_link* is the last composition or existence link to *origin*:
OBJECT_IS_IN_USE_FOR_DELETE (*origin*)

USAGE_MODE_ON_LINK_TYPE_WOULD_BE_VIOLATED (*origin*, *link*, DELETE_MODE)

9.2.3 LINK_DELETE_ATTRIBUTE

```
LINK_DELETE_ATTRIBUTE (
    origin      : Object_designator,
    link        : Link_designator,
    attribute   : Attribute_designator
)
```

LINK_DELETE_ATTRIBUTE deletes the non-key attribute *attribute* of the link *link* of the object *origin*, if the "attribute_type" object representing the attribute type of *attribute* is no longer in the object base.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

A write lock of the default mode is obtained on *link*. A read lock of the default mode is obtained on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*origin*, ATOMIC, MODIFY, WRITE_LINKS)
PRIVILEGE_IS_NOT_GRANTED (PCTE_CONFIGURATION)
REFERENCED_OBJECT_IS_NOT_MUTABLE (key of *link*)

NOTE - It is the responsibility of the user to ensure that the attribute type is no longer in the object base.

9.2.4 LINK_GET_ATTRIBUTE

```
LINK_GET_ATTRIBUTE (
    origin      : Object_designator,
    link        : Link_designator,
    attribute   : Attribute_designator
)
    result      : Attribute_value
```

LINK_GET_ATTRIBUTE returns the value of the non-key attribute *attribute* of the link *link* of the object *origin*.

A read lock of the default mode is obtained on *link*. A read lock of the default mode is obtained on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*origin*, ATOMIC, READ, READ_LINKS)
USAGE_MODE_ON_ATTRIBUTE_TYPE_WOULD_BE_VIOLATED (*origin*, *link*, *attribute*, READ_MODE)

9.2.5 LINK_GET_DESTINATION_VOLUME

```
LINK_GET_DESTINATION_VOLUME (
    origin          : Object_designator,
    link            : Link_designator
)
    destination     : Volume_info
```

LINK_GET_DESTINATION_VOLUME returns the volume identifier *volume_identifier* and the volume accessibility *mounted* of the volume on which the destination *dest* of the link *link* of the object *origin* resides. The returned value of *mounted* is as follows:

- ACCESSIBLE if the volume on which *dest* resides is mounted and is accessible in the network partition that contains the calling procedure's workstation. In this case, *volume_identifier* is the volume identifier of the volume on which *dest* resides.
- INACCESSIBLE if the PCTE implementation is able to determine on which volume the object resides, and that volume is not accessible (either because the volume is not mounted or because the volume is mounted in a network partition which does not contain the calling process's workstation). In this case, *volume_identifier* is the volume identifier of the volume on which the object resides.
- UNKNOWN if the PCTE implementation is unable to determine on which volume the object resides. In this case, *volume_identifier* is the volume identifier of a volume which is not currently accessible.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

The situations in which UNKNOWN is returned rather than INACCESSIBLE, and vice versa, are implementation-defined, as is the general meaning of the volume identifier returned for UNKNOWN. In any particular situation, the choice is implementation-dependent.

Read locks of the default mode are obtained on *link*, and on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*origin*, ATOMIC, READ, READ_LINKS)

LINK_DESTINATION_DOES_NOT_EXIST (*link*)

OBJECT_IS_ARCHIVED (destination of *link*)

USAGE_MODE_ON_LINK_TYPE_WOULD_BE_VIOLATED (*origin*, *link*, NAVIGATE_MODE)

NOTE - Some implementations may be able to guarantee that only ACCESSIBLE or INACCESSIBLE is returned. For implementations that return UNKNOWN, the volume identifier returned should be that of the volume which should be made accessible prior to repeating the call. The destination object may reside on this volume, or it may contain implementation-dependent details of the volume on which the object resides, or of a further volume to be made accessible. Although the operation may require access to other volumes, no error condition is raised as a result.

9.2.6 LINK_GET_KEY

```
LINK_GET_KEY (
    origin   : Object_designator,
    link     : Link_designator
)
    key      : [ Actual_key ]
```

LINK_GET_KEY returns in *key* the complete sequence of key attribute values (if any) of the link *link* of the object *origin*.

Read locks of default mode are obtained on *link*, and on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*origin*, ATOMIC, READ, READ_LINKS)

USAGE_MODE_ON_LINK_TYPE_WOULD_BE_VIOLATED (*origin*, *link*, NAVIGATE_MODE)

9.2.7 LINK_GET_REVERSE

```
LINK_GET_REVERSE (
    origin          : Object_designator,
    link            : Link_designator
)
    reverse_link    : [ Link_designator ],
    dest            : Object_designator
```

LINK_GET_REVERSE returns in *reverse_link* the reverse link (if there is one) of the link *link* of the object *origin*, and in *dest* the destination of *link*. If *link* has no reverse link, no value is returned in *reverse_link*.

Read locks of the default mode are obtained on *link*, and on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*origin*, ATOMIC, READ, READ_LINKS)

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

ACCESS_ERRORS (destination of *link*, ATOMIC, READ, READ_LINKS)
LINK_DESTINATION_DOES_NOT_EXIST (*link*)
LINK_NAME_IS_TOO_LONG_IN_CURRENT_WORKING_SCHEMA
REFERENCE_CANNOT_BE_ALLOCATED
USAGE_MODE_ON_LINK_TYPE_WOULD_BE_VIOLATED (*origin*, *link*, NAVIGATE_MODE)

9.2.8 LINK_GET_SEVERAL_ATTRIBUTES

```
LINK_GET_SEVERAL_ATTRIBUTES (
    origin      : Object_designator,
    link        : Link_designator,
    attributes  : Attribute_selection
)
    values      : Attribute_assignments
```

LINK_GET_SEVERAL_ATTRIBUTES returns in *values* a set of attribute assignments of the link *link* of the object *origin*.

The returned set of attributes is determined by *attributes*:

- a set of attribute designators: the set of non-key attributes, as for LINK_GET_ATTRIBUTE (*origin*, *link*, A) for each attribute A of *attributes*;
- VISIBLE_ATTRIBUTE_TYPES: all non-key attributes of *link* visible in the working schema of the calling process and with usage mode including READ_MODE.

Read locks of the default mode are obtained on *link*, and on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*origin*, ATOMIC, READ, READ_LINKS)
If *attributes* is not VISIBLE_ATTRIBUTE_TYPES:
USAGE_MODE_ON_ATTRIBUTE_TYPE_WOULD_BE_VIOLATED (*origin*, *link*, each element of *attributes*, READ_MODE)

9.2.9 LINK_REPLACE

```
LINK_REPLACE (
    origin              : Object_designator,
    link                : Link_designator,
    new_origin          : Object_designator,
    new_link            : Link_designator,
    new_reverse_key     : [ Actual_key ]
)
```

LINK_REPLACE replaces the composition or existence link *link* of the object *origin* by a new composition or existence link as specified by *new_link* from *new_origin*, with the same destination *dest*.

The new link from *new_origin* to *dest* is created and *link* is deleted in the same way as the following sequence of operations, ignoring any temporary violation of link bounds or composition exclusivity on *dest*:

```
LINK_CREATE (new_origin, new_link, dest, new_reverse_key);
LINK_DELETE (origin, link)
```

Write locks of the default mode are obtained on the links to be deleted and on the new links. A read lock of the default mode is obtained on *new_origin* if the interpretation of *link* or *new_link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7). A read lock of the

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

default mode is obtained on *dest* if the interpretation of *new_reverse_key* implies the evaluation of any '+' or '++' key attribute values.

A write lock of the default mode is obtained on *dest* and each of its components if the OWNER discretionary access right is granted or denied for one or more groups to *new_origin*, and different OWNER discretionary access rights exist for one or more of those same groups to *dest*.

Errors

ACCESS_ERRORS (*new_origin*, ATOMIC, MODIFY, APPEND_LINKS)

ACCESS_ERRORS (*origin*, ATOMIC, MODIFY, WRITE_LINKS)

If the reverse link of *new_link* is implicit:
ACCESS_ERRORS (*dest*, ATOMIC, CHANGE, APPEND_IMPLICIT)

If the reverse link of *new_link* is not implicit:
ACCESS_ERRORS (*dest*, ATOMIC, MODIFY, APPEND_LINKS)

If the reverse link of *link* is implicit:
ACCESS_ERRORS (*dest*, ATOMIC, CHANGE, WRITE_IMPLICIT)

If the reverse link of *link* is not implicit:
ACCESS_ERRORS (*dest*, ATOMIC, MODIFY, WRITE_LINKS)

If *new_link* is atomically stabilizing and *link* is not, or vice versa:
ACCESS_ERRORS (*dest*, ATOMIC, CHANGE, STABILIZE)

If *new_link* is compositely stabilizing and *link* is not, or vice versa:
ACCESS_ERRORS (*dest*, COMPOSITE, CHANGE, STABILIZE)

CATEGORY_IS_BAD (*new_origin, new_link*, (COMPOSITION, EXISTENCE))

CATEGORY_IS_BAD (*origin, link*, (COMPOSITION, EXISTENCE))

COMPONENT_ADDITION_ERRORS (*dest, new_link*)

DESTINATION_OBJECT_TYPE_IS_INVALID (*new_origin, new_link, dest.*)

DESTINATION_OBJECT_TYPE_IS_INVALID (*origin, link, dest*)

If *new_link* is of category COMPOSITION, and *new_origin* has OWNER granted or denied:
LINK_EXISTS (*new_origin, new_link*)

OBJECT_CANNOT_BE_STABILIZED (*dest*)

LOWER_BOUND_WOULD_BE_VIOLATED (*origin, link*)

REVERSE_KEY_IS_BAD (*new_origin, new_link, new_reverse_key*)

REVERSE_KEY_IS_NOT_SUPPLIED (*new_origin, new_link, dest*)

REVERSE_LINK_EXISTS (*new_origin, new_link, dest, new_reverse_key*)

UPPER_BOUND_WOULD_BE_VIOLATED (*dest*, reverse link of *new_link*)

UPPER_BOUND_WOULD_BE_VIOLATED (*new_origin, new_link*)

USAGE_MODE_ON_LINK_TYPE_WOULD_BE_VIOLATED (*origin, link*, DELETE_MODE)

USAGE_MODE_ON_LINK_TYPE_WOULD_BE_VIOLATED (*new_origin, new_link*, CREATE_MODE)

NOTE - *new_reverse_key* can be supplied in particular to replace an exclusive link or a link whose reverse link is of cardinality one.

9.2.10 LINK_RESET_ATTRIBUTE

```
LINK_RESET_ATTRIBUTE (
    origin      : Object_designator,
    link        : Link_designator,
    attribute   : Attribute_designator
)
```


Documents
Document Sharing Hub
Full document details: ISO_IEC 13719-1_1995

LINK_RESET_ATTRIBUTE resets the non-key attribute *attribute* of the link *link* of the object *origin* to its initial value.

A write lock of the default mode is obtained on *link*. A read lock of the default mode is obtained on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*object*, ATOMIC, MODIFY, WRITE_ATTRIBUTES)

KEY_UPDATE_IS_FORBIDDEN (*attribute*)

REFERENCED_OBJECT_IS_NOT_MUTABLE (key of *link*)

USAGE_MODE_ON_ATTRIBUTE_TYPE_WOULD_BE_VIOLATED (*origin*, *link*, *attribute*, WRITE_MODE)

9.2.11 LINK_SET_ATTRIBUTE

```
LINK_SET_ATTRIBUTE (
    origin      : Object_designator,
    link        : Link_designator,
    attribute   : Attribute_designator,
    value       : Attribute_value
)
```

LINK_SET_ATTRIBUTE assigns the value *value* to the non-key attribute *attribute* of the link *link* of the object *origin*.

A write lock of the default mode is obtained on *link*. A read lock of the default mode is obtained on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*origin*, ATOMIC, MODIFY, WRITE_LINKS)

ENUMERATION_VALUE_IS_OUT_OF_RANGE (*value*, values of *attribute*)

KEY_UPDATE_IS_FORBIDDEN (*origin*, *link*, *attribute*)

If *link* is a "referenced_object" link:
REFERENCED_OBJECT_IS_NOT_MUTABLE (key of *link*)

USAGE_MODE_ON_ATTRIBUTE_TYPE_WOULD_BE_VIOLATED (*origin*, *link*, *attribute*, WRITE_MODE)

VALUE_LIMIT_ERRORS (*value*)

The following implementation-dependent error may be raised:
VALUE_TYPE_IS_INVALID (*value*, *origin*, *link*, *attribute*)

9.2.12 LINK_SET_SEVERAL_ATTRIBUTES

```
LINK_SET_SEVERAL_ATTRIBUTES (
    origin      : Object_designator,
    link        : Link_designator,
    attributes  : Attribute_assignments
)
```

For each element A in the domain of *attributes*, LINK_SET_SEVERAL_ATTRIBUTES sets the value of the attribute A of the link *link* of the object *origin* to the value *attributes* (A), in the same way as:

LINK_SET_ATTRIBUTE (*origin*, *link*, A, *attributes* (A))

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

A write lock of the default mode is obtained on *link*. A read lock of the default mode is obtained on *origin* if the interpretation of *link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

Errors

ACCESS_ERRORS (*origin*, ATOMIC, MODIFY, WRITE_LINKS)

For each element A in the domain of *attributes*:

ENUMERATION_VALUE_IS_OUT_OF_RANGE (*attribute*(A), values of A)
KEY_UPDATE_IS_FORBIDDEN (*origin*, *link*, A)
USAGE_MODE_ON_ATTRIBUTE_TYPE_WOULD_BE_VIOLATED (*origin*, *link*, A, WRITE_MODE)
VALUE_LIMIT_ERRORS (*attributes* (A))

The following implementation-dependent error may be raised for each element A in the domain of *attributes*:

VALUE_TYPE_IS_INVALID (*attributes* (A), *origin*, *link*, A)

9.3 Object operations

9.3.1 OBJECT_CHECK_TYPE

```
OBJECT_CHECK_TYPE(
    object      : Object_designator,
    type2       : Object_type_nominator
)
    relation    : Type_ancestry
```

OBJECT_CHECK_TYPE compares the object type *type1* of the object *object* against the object type *type2*, and returns in *relation* a value defined as follows:

- EQUAL_TYPE if *type1* is the same as *type2*;
- ANCESTOR_TYPE if *type1* is an ancestor of *type2*;
- DESCENDANT_TYPE if *type1* is a descendant type of *type2* in the working schema of the calling process;
- UNRELATED_TYPE in all other cases.

The visibility of the type of *object* does not affect the result of the operation.

A read lock of the default mode is obtained on *object*.

Errors

CONFIDENTIALITY_WOULD_BE_VIOLATED (*object*, ATOMIC)
INTEGRITY_CONFINEMENT_WOULD_BE_VIOLATED (*object*, ATOMIC)
OBJECT_IS_ARCHIVED (*object*)
OBJECT_TYPE_IS_UNKNOWN (*type2*)
VOLUME_IS_INACCESSIBLE (*object*, ATOMIC)

9.3.2 OBJECT_CONVERT

```
OBJECT_CONVERT(
    object  : Object_designator,
    type    : Object_type_nominator
)
```

OBJECT_CONVERT changes the object type of the object *object* to the descendant object type *type*.

AVA Documents Document Sharing Hub
Full document details: ISO_IEC 13719-1_1995

The operation has no effect if the current type of *object* is already *type*.

A write lock of the default mode is obtained on *object*.

Errors

ACCESS_ERRORS (*object*, ATOMIC, CHANGE, CONTROL_OBJECT)
OBJECT_IS_STABLE (*object*)
OBJECT_TYPE_IS_INVALID (*type*)
OBJECT_TYPE_IS_UNKNOWN (*type*)
TYPE_IS_NOT_DESCENDANT (object type of *object*, *type*)
If *object* is not of type *type*:
USAGE_MODE_ON_OBJECT_TYPE_WOULD_BE_VIOLATED (current type of object, *type*)

9.3.3 OBJECT_COPY

```
OBJECT_COPY (
    object              : Object_designator,
    new_origin          : Object_designator,
    new_link            : Link_designator,
    reverse_key         : [ Actual_key ],
    on_same_volume_as   : [ Object_designator ],
    access_mask         : Atomic_access_rights
)
    new_object          : Object_designator
```

OBJECT_COPY creates an object *new_object* as a *copy* of *object*. More precisely:

- If *object* is a file, an accounting log, or an audit file, then the contents of *new_object* is the same as the contents of *object*; if *object* is a pipe then the contents of *new_object* is empty.
- For each duplicable attribute X of *object*, there is an attribute of *new_object* which is a copy of X.
- For each duplicable direct component X of *object*, there is a direct component of *new_object* which is a copy of X.
- For each duplicable internal link A of *object* whose destination is a duplicable component, there is an internal link B of *new_object* such that the destination of B is the copy of the destination of A and all other properties of B are the same as for A.
- For each duplicable external link A of *object*, there is a corresponding external link of *new_object* which is a copy of A.
- For each non-duplicable attribute of *object*, there exists a corresponding attribute of *new_object* whose value is either set to the initial value of the attribute type or, for the following predefined attributes: the exact identifier, volume identifier, replicated state, contents type, last access time, last modification time, last change time, last composite access time, last composite modification time, last composite change time, number of incoming links, number of incoming composition links, number of incoming existence links, number of incoming reference links, number of incoming stabilizing links, number of outgoing composition links, number of outgoing existence links, atomic ACL, composite ACL, confidentiality label, and integrity label, is set to a value corresponding to the newly created object. In particular, since the attribute "replicated_state" is not copied, the copy of a replicated object is not replicated.
- For each non-duplicable non-key attribute of a copied link, there exists a corresponding non-key attribute of the copied link whose value is set to the initial value of the attribute type.

A *copy* of an attribute A is an attribute which has the same attribute type, attribute value, and attribute properties as A.

AVA Documents Document Sharing Hub

Full document details: ISO_IEC 13719-1_1995

A *copy* of a link A is a link which has the same link type, key attributes, duplicable non-key attributes, link properties, and destination as A.

A *copy* of a component X of an object A is a component Y of a copy B of A such that Y is a copy of X as an object, and the composition link from B to Y is a copy of the composition link from X to A

OBJECT_COPY creates a link *link* of the object *new_origin*, as specified by *new_link*, and with *new_object* as destination, and its reverse link *reverse_link* with origin *new_object* and key derived from *reverse_key* as described in 23.1.2.7.

access_mask is used in conjunction with the default atomic ACL and default object owner of the calling process to specify the atomic ACL and the composite ACL of *new_object* and its components. See 19.1.4 for more details.

If *new_link* is a composition link, then any security group that has OWNER granted or denied to *new_origin* has OWNER granted or denied respectively to *dest*; similarly if *reverse_link* is a composition link, then any security group that has OWNER granted or denied to *dest* has OWNER granted or denied respectively to *new_origin*.

If *new_link* is a composition link, then any security group that has OWNER granted or denied to *origin* has OWNER granted or denied respectively to *dest*; similarly if *reverse_link* is a composition link, then any security group that has OWNER granted or denied to *dest* has OWNER granted or denied respectively to *origin*.

new_object has the same integrity and confidentiality label as *object* and each component of *new_object* has the same integrity and confidentiality labels as the corresponding component of *object*.

If *on_same_volume_as* is supplied, *new_object* resides on the same volume as the object *on_same_volume_as*. Otherwise, *new_object* resides on the same volume as *object* and each component of *new_object* resides on the same volume as its corresponding component in *object*.

An "object_on_volume" link is created from the volume on which *new_object* resides to *new_object*, and similarly for all its components. Each created link is keyed by the exact identifier of its destination object.

Read locks of the default mode are obtained on *object* and on all its components; write locks of the default mode are obtained on the new objects and links except the new "object_on_volume" links. A read lock of the default mode is obtained on *new_object* if the interpretation of the link *new_link* implies the evaluation of any '+' or '++' key attribute values (see 23.1.2.7).

If *object* is an accounting log, its contents is preserved.

Errors

ACCESS_ERRORS (*new_origin*, ATOMIC, MODIFY, APPEND_LINKS)
ACCESS_ERRORS (*object*, COMPOSITE, READ, READ_LINKS)
ACCESS_ERRORS (*object*, COMPOSITE, READ, READ_ATTRIBUTES)
ACCESS_ERRORS (*object*, COMPOSITE, READ, READ_CONTENTS)
ACCESS_ERRORS (*on_same_volume_as*, ATOMIC, SYSTEM_ACCESS)
If *new_link* is compositely stabilizing:
    ACCESS_ERRORS (*new_object*, COMPOSITE, CHANGE, STABILIZE)
If *reverse_link* is compositely stabilizing:
    ACCESS_ERRORS (*new_origin*, COMPOSITE, CHANGE, STABILIZE)
For each destination X of a duplicable external link L of a duplicated component:
    ACCESS_ERRORS (X, ATOMIC, CHANGE, APPEND_IMPLICIT)
    If L is atomically stabilizing:
        ACCESS_ERRORS (X, ATOMIC, CHANGE, STABILIZE)

AVA Documents Document Sharing Hub
Full document details: ISO_IEC 13719-1_1995