INTERNATIONAL STANDARD

**ISO/IEC 13719-3**

Second edition
1998-10-01

# Information technology — Portable Common Tool Environment (PCTE) —

## Part 3:
Ada programming language binding

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

*Partie 3: Liant de langage de programmation Ada*



Reference number
ISO/IEC 13719-3:1998(E)

## Contents

## 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 13719-3 was prepared by ECMA (as Standard ECMA-162) 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.

This second edition cancels and replaces the first edition (ISO/IEC 13719-3:1995), which has been technically revised.

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*

– *Part 4: IDL binding (Interface Definition Language)*

Annex A forms an integral part of this part of ISO/IEC 13719.

# Information technology — Portable Common Tool Environment (PCTE) —

# Part 3:
Ada programming language binding

## 1 Scope

This part of ISO/IEC 13719 defines the binding of the Portable Common Tool Environment (PCTE) interfaces, as specified in ISO/IEC 13719-1, to the Ada programming language.

A number of features are not completely defined in ISO/IEC 13719-1, some freedom being allowed to the implementor. Some of these features are specified as implementation limits. Some constraints are placed on these implementation limits by this part of ISO/IEC 13719. These constraints are specified in clause 24.

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 system development, are often referred to as tools.

## 2 Conformance

An implementation of PCTE conforms to this part of ISO/IEC 13719 if it conforms to 2.2 of ISO/IEC 13719-1, where the binding referred to there is taken to be the Ada language binding defined in clauses 1 to 5 and 8 to 25 of this part of ISO/IEC 13719. All other parts of this part of ISO/IEC 13719 are provided as assistance to the reader and are not normative.

The Ada language binding defined in this part of ISO/IEC 13719 conforms to 2.1 of ISO/IEC 13719-1.

## 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 part of ISO/IEC 13719 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 13719-1:1998, | *Information technology - Portable Common Tool Environment (PCTE) - Part 1: Abstract specification.* |
| ISO/IEC 13719-2:1998, | *Information technology - Portable Common Tool Environment (PCTE) - Part 2: C programming language binding.* |
| ISO 8601:1988, | *Data elements and interchange formats - Information interchange - Representation of dates and times.* |
| ISO/IEC 8652:1995, | *Information technology - Programming languages - Ada.* |

ISO/IEC TR 10182:1993, *Information technology - Programming languages, their environments and system software interfaces - Guidelines for language bindings.*

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

## 4 Definitions

All technical terms used in this part of ISO/IEC 13719, other than a few in widespread use, are defined in the body of this part of ISO/IEC 13719 or in the referenced documents.

## 5 Formal notations

All datatypes and subprogram definitions are expressed using ISO/IEC 8652 conformant syntax.

## 6 Outline of the Standard

Clause 7 describes the strategy used to develop this binding specification.

Clause 8 defines the mapping from the datatypes that are used in the abstract specification to Ada programming language datatypes.

Clauses 9 to 22 define the bindings of datatypes and operations in the corresponding clauses of ISO/IEC 13719-1. The extensions for fine-grain objects are added at the end of clause 11.

Clause 23 defines the binding of object and type references, as specified in ISO/IEC 13719-1 23.1.2 and 23.2. Because of the package structure, this clause consists of a cross-reference to the definitions which are in 8.4.

Clause 24 defines the binding of the implementation limit subprograms described in ISO/IEC 13719-1, clause 24.

Clause 25 defines the binding of the error conditions specified in ISO/IEC 13719-1, clause 25, and defines binding-defined error conditions for the Ada binding.

Annex A, which is normative, contains the extensions for object orientation, corresponding to annex G of ISO/IEC 13719-1.

## 7 Binding strategy

### 7.1 Ada programming language standard

The Ada package specifications were designed to conform to ISO/IEC 8652.

### 7.2 General principles

The following general principles were applied when generating the binding in this part of ISO/IEC 13719.

ISO/IEC TR 10182 should be followed as far as possible for binding method 1: provide a completely defined procedural interface.

Each operation in ISO/IEC 13719-1 should be represented by one subprogram in this part of ISO/IEC 13719 unless there are specific reasons to the contrary.

All Ada identifiers should be in lower case except for predefined identifiers, named constant values, and enumeration literals. Since the Ada standard is insensitive to case this is for typographical consistency between ISO/IEC 13719-1, ISO/IEC 13719-2, and this part of ISO/IEC 13719.

Nondefining occurrences of the names of Ada subprograms and types should use the fully qualified form, so as to identify all package dependences.

All the Ada packages should have names that begin with 'Pcte_' to ensure they are unique within an Ada Library System. The choice of case of the characters of 'Pcte' is for typographical consistency with ISO/IEC 13719-2.

An abstract operation with name of the form 'TYPE_VERB_PHRASE' should be mapped to an Ada subprogram 'verb_phrase' declared by a package called 'Pcte_type'. For example, 'PROCESS_SET_WORKING_SCHEMA' is mapped to 'Pcte_process.set_working_schema'

When a package hierarchy is required, it should be compatible with the abstract specification clause organisation. For example, 'ACCOUNTING_LOG_READ' is mapped to 'Pcte_accounting.log.read'.

Names should be retained from ISO/IEC 13719-1 as far as possible.

All additional names should be chosen appropriately for their meanings.

Each operation that can return errors should have an additional **in** parameter of an access type designating an object into which error indications can be returned. This allows the subprograms to be procedures or functions as appropriate.

Wherever practical, types introduced for passing complex data entities between a caller and a subprogram should be private or limited private. Limited private types should be used unless the basic operations on entities of such types are safe and consistent with ISO/IEC 13719-1.

All simple parameter types in ISO/IEC 13719-1 that represent attribute value types should be mapped to corresponding Ada types defined by this binding.

All simple parameter types in ISO/IEC 13719-1 that do not represent attribute value types should be mapped to predefined types or subtypes or derived types of predefined types.

## 7.3 Dynamic memory management

A type defined in this part of ISO/IEC 13719 for which an object is created dynamically is always limited private, and subprograms are provided to construct, access and discard such objects.

It is the responsibility of the Ada program that declares a variable of such an Ada type to ensure that its 'construct' subprogram is called before the variable is read; the construct subprogram always initializes the value of the variable. Use of a variable of this type before calling its construct subprogram, or after calling its discard subprogram always causes an error to be generated.

It is the responsibility of the Ada program that defines variables of such Ada types to ensure that the 'discard' subprogram is called when the variable is no longer needed and before the immediate scope of the variable is left. If such a variable goes out of scope without being discarded there is the possibility that the memory allocated to it cannot be recovered, which may result in the exception STORAGE_ERROR being raised subsequently.

If a constructed variable is constructed again without having been discarded, no error is reported; such usages of the interface are regarded as akin to allowing such variables to go out of scope before they are discarded.

## 7.4 Complex entities as parameters

Some complex entities to be passed into or retrieved from an operation are defined as sets or sequences of a base type in ISO/IEC 13719-1. Where such sets and sequences are bounded and constrainable by the user they are mapped to Ada language array types.

Where such sets and sequences are unbounded they are mapped to limited private types declared by nested packages. These limited private types are defined with operations for construction with initial value, access and discarding. A value is assigned to an object of one of these types either as a parameter of mode **out** or **in out** of a subprogram corresponding to an abstract operation of ISO/IEC 13719-1, or by use of a subprogram of the appropriate nested package.

Thus the data values contained in sequences can be easily manipulated using Ada language facilities, while allowing the implementation to choose the best implementation.

All the operations that have a sequence as an **in out** parameter create the elements of the sequence and return it as a result of the operation. The user does not need to create the elements of the sequence in advance: the user needs only to declare a sequence variable and pass that variable.

## 7.5 Character strings

Values of datatype String are manipulated as the Ada type Pcte.string which is a subtype of the predefined type STANDARD.STRING.

## 7.6 Error conditions

This part of ISO/IEC 13719 allows any error to be recorded as an error value or to be raised as an exception. This is achieved by providing a limited private type, Pcte_error.handle, into objects of which type error values, established by a subprogram defined in this part of ISO/IEC 13719, can be recorded. These handles may also be set to cause any error to be raised as an exception instead of recording it as an error value.

## 7.7 Implementation limits

ISO/IEC 13719-1 defines a set of limits that must be honoured by all implementations of a Language Binding. Clause 24 defines the Ada names for these limits and the operations by which they can be retrieved.

# 8 Datatype mapping

This clause defines the mapping of the parameter and result datatypes of the operations of ISO/IEC 13719-1 (*PCTE datatypes*) to the parameter and result datatypes of the operations of this part of ISO/IEC 13719 (*Ada datatypes*).

PCTE datatype names are printed in normal characters, the names of parameters to operations of ISO/IEC 13719-1 are printed in italics and the names of the operations themselves are printed in all upper case characters. PCTE datatype constructors are printed in bold. LI datatype names are printed in italics. Ada datatype names are printed in normal characters, as are the names of Ada subprograms and their parameter names. Enumeration literals, exception names and the names of constant objects are printed in all upper case characters. Ada reserved words are printed in bold.

The mapping from PCTE datatypes to Ada datatypes is done in two stages, via *LI datatypes* defined in ISO/IEC DIS 11404.

## 8.1 Mapping of PCTE datatypes to LI datatypes

As far as possible the names of PCTE datatypes are retained, with minor typographical changes, for the corresponding LI datatypes, but some new names are introduced.

The general strategy of this mapping is as follows.

- To select for each PCTE datatype a LI datatype definition which matches the requirements of the PCTE datatype defined in ISO/IEC 13719-1. The LI datatype definition is, where possible, a primitive LI datatype, and otherwise a generated LI datatype.
- To define new datatype generators where needed.
- To map PCTE datatypes with the same properties to the same LI datatype.

### 8.1.1 Mapping of predefined PCTE datatypes

The mapping of these PCTE datatypes is as defined in ISO/IEC 13719-1, clause 23 and is summarized in table 1.

**Table 1 - Mapping of attribute value types**

| PCTE datatype | LI datatype |
|---|---|
| Boolean | *boolean* |
| Integer | *pcte-integer = integer range (lb1 .. ub1)* |
| Natural | *pcte-natural = integer range (0 .. ub1)* |
| Float | *pcte-float = real (10, f1) range (lb2 .. ub2)* |
| Time | *pcte-time = time (second, 10, f2) range (lb3 .. ub3)* |
| Octet | *octet* |
| Text | *pcte-text = characterstring (repertoire)* |
| Enumerated type xxx=VALUE1\|VALUE2\|... | *pcte-xx = enumerated (value1, value2, ...)* |

### 8.1.2 Mapping of private PCTE datatypes

**Table 2 - Mapping of other primitive PCTE datatypes**

| PCTE datatype | LI datatype |
|---|---|
| Address | *see below* |
| Contents_handle | *contents-handle = private* |
| Handler | *not used* |
| Message_handle | *not used* |
| Message_type | *see below* |
| Object_reference | *object-reference = private* |
| Link_reference | *link-reference = private* |
| Type_reference | *type-reference = private* |
| Position_handle | *position-handle = private* |
| Profile_handle | *profile-handle = private* |



The PCTE datatype Address is mapped directly to a subtype of the predefined Ada type SYSTEM.ADDRESS.

The PCTE datatype Message_type is mapped directly to an Ada record type Pcte_message.message_type.

### 8.1.3 Mapping of complex PCTE datatypes

PCTE sequence datatypes are mapped via the new datatype generator *pcte-sequence* (see 8.1.4).

PCTE set datatypes are divided into bounded set types and unbounded set types. Bounded set types have values which are sets of enumeration values with at most 32 possible elements; all others are unbounded set types. Bounded set types are mapped via the new LI datatype generator *Bounded-set*. Unbounded set types are mapped via the new LI datatype generator *pcte-sequence*; the order of elements in the sequence is ignored by the operation. For Message_types see 14.1.

When used as input parameter of an operation in clauses 9 to 22, a sequence which represents a PCTE unbounded set may contain repeated elements. The effect for the operation is as though each element occurred only once.

When returned as the result of an operation in clauses 9 to 22, an unbounded set has no repeated elements. The order of the elements in the sequence is arbitrary except for the procedure normalize, see 8.4.

PCTE map datatypes are notionally mapped via a new LI datatype generator *Map*; their mappings to Ada datatypes are defined directly (see 19.1).

PCTE union datatypes other than enumerations are notionally mapped via the datatype generator *Choice*. The only such PCTE datatypes are auditing-record, accounting-record and value-type, their mappings to Ada datatypes are defined directly (see 21.1, 22.1, 9.1 respectively).

PCTE composite and product datatypes (except composite datatypes with a single Token field, for such see 8.1.2) are mapped to the datatype generator *Record*. The component types of the composite or product type are mapped as defined by the rules of 8.1. The PCTE datatype Message is treated specially, see 14.1.

### 8.1.4 New LI datatype generators

**Pcte-sequence**

Description: *Pcte-sequence* is a datatype generator derived from *Sequence* by adding further characterizing operations. The values of a pcte-sequence, called its elements, are indexed sequentially from 1.

pcte-sequence of (*base*) = new sequence of (*base*)

The characterizing operations are: Equal, IsEmpty, Head, Tail, Empty and Append from *Sequence*, plus Get, Put, Delete, InsertSequence, AppendSequence, LengthOf, IndexOf, and Normalize.

Get (s : sequence of *base*, i : ordinal) : *base*
is the element of s with index i.

Put s : sequence of *base*, e : *base*, i : ordinal) : *base*
is the sequence formed from s by inserting e as an element of s immediately before the element with index i.

Delete (s : sequence of *base*, i : ordinal) : sequence of *base*
is the sequence formed from s by deleting the element with index i.

InsertSequence (s1, s2 : sequence of *base*, i : ordinal) : sequence of *base*
is the sequence formed from s1 by inserting the sequence s2 immediately before the element with index i.

AppendSequence (s1, s2 : sequence of *base*) : sequence of *base*
is the sequence formed from s1 by appending the sequence s2 at the end.

LengthOf (s : sequence of *base*) : natural
is the number of elements of s.

IndexOf (s : sequence of *base*, e : *base*) : natural
is the index of the first occurrence of the element e in the sequence s if the element is a member of the sequence, and 0 otherwise.

Normalize (s : sequence of *base*) : sequence of *base*
is the sequence formed from s by ordering the elements in an implementation-defined canonical order and deleting duplicate elements.

**Bounded-set**

Description: Bounded-set is a datatype generator derived from *Set* by restricting the cardinality of the values to 32 or less.

bounded-set of *base* = new set of (*base*) : size (0 .. 32)

The characterizing operations are: IsIn, Subset, Equal, Difference, Union, Intersection, Empty, SetOf, Select from *Set*.

## 8.2 Mapping of LI datatypes to Ada datatypes

### 8.2.1 LI datatype: boolean

The LI datatype *boolean* is mapped to the boolean Ada datatype Pcte.boolean.

**subtype** boolean **is** STANDARD.BOOLEAN

**Characterizing operations**

| **Operation** | **Ada Operation** |
|---|---|
| Equal (x,y) | x = y |
| Not (x) | **not** x |
| And (x,y) | x **and** y |
| Or (x,y) | x **or** y |

### 8.2.2 LI datatype: pcte-integer

The LI datatype *pcte-integer* is mapped to the integer Ada datatype Pcte.integer.

**type** integer **is**
**range** MIN_INTEGER_ATTRIBUTE .. MAX_INTEGER_ATTRIBUTE;

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| Equal (x, y) | x = y |
| Add (x, y) | x + y |
| Multiply (x, y) | x * y |
| Negate (x) | -x |
| NonNegative (x) | x >= 0 |
| InOrder (x, y) | x <= y |

### 8.2.3 LI datatype: pcte-natural

The LI datatype *pcte-natural* is mapped to the integer Ada datatype Pcte.natural.

**type** natural **is range** 0 .. Pcte.integer'LAST;

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| Equal (x, y) | x = y |
| Add (x, y) | x + y |
| Multiply (x, y) | x * y |
| InOrder (x, y) | x <= y |

### 8.2.4 LI datatype: pcte-float

The LI datatype *pcte-float* is mapped to the Ada datatype Pcte.float.

**type** float **is digits** MAX_DIGITS_FLOAT_ATTRIBUTE
**range** MIN_FLOAT_ATTRIBUTE .. MAX_FLOAT_ATTRIBUTE;

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| Equal (x,y) | x = y |
| Add (x, y) | x + y |
| Multiply (x, y) | x * y |
| Negate (x) | -x |
| Reciprocal (x) | 1.0/x |
| NonNegative (x) | x >= 0.0 |
| InOrder (x,y) | x <= y |

### 8.2.5 LI datatype: pcte-time

The LI datatype *pcte-time* is mapped to the Ada datatype Pcte.calendar.time.

**type** time **is private**;

The range and accuracy of Pcte.calendar.time is implementation-defined but must respect the conditions for conformance with ISO/IEC 13719-1. Conversions are supported between CALENDAR.TIME and Pcte.calendar.time. A specific accuracy (year, month, day, hour, minute, second, fractions of a second) is obtained by using such components of Pcte.calendar as are required through the functions year, month, day, seconds, and time_of, and the procedure split in the Ada package Pcte.calendar, together with appropriate arithmetic for the determination of hours, minutes and seconds.

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| Equal (x,y) | x = y |
| InOrder (x,y) | x <= y |
| Difference (x,y) | x - y |
| Extend.*res1tores2* (x) | Pcte.calendar.extend (x) |
| Round.*res1tores2* (x) | Pcte.calendar.round (x) |

### 8.2.6 LI datatype: octet

The LI datatype *octet* is mapped to the Ada datatype STANDARD.CHARACTER.

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| Equal (x,y) | x = y |

### 8.2.7 LI datatype: pcte-text

The LI datatype *pcte-text* is mapped to the Ada datatype Pcte.text.

```
subtype text is STANDARD.STRING;
```

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| Head (s) | s(s'FIRST) |
| Tail (s) | s(s'FIRST+1 .. s'LAST) |
| Equal (s1, s2) | s1 = s2 |
| Empty (s) | "" |
| Append (s, c) | s & c |
| IsEmpty (s) | s'LENGTH = 0 |

### 8.2.8 LI datatype generator: pcte-sequence

The LI datatypes of the family *pcte-sequence* are mapped to types declared by Ada packages. In general, each LI datatype *sequence of xxx* is mapped to a package called yyys (the plural of yyy) where yyy is the mapping of *xxx*, e.g. for an arbitrary type named element:

```
package elements is

   type sequence is limited private;

   -- The initial value of an object of type sequence is an empty sequence, i.e. one of
   -- length 0.  To discard a sequence so that the associated storage can be recovered,
   -- all the elements of the sequence are deleted by means of procedure discard.

   function get (
      list      : elements.sequence;
      index     : Pcte.natural := Pcte.natural'FIRST;
      status    : Pcte_error.handle := EXCEPTION_ONLY)
      return     element;

   -- elements.get returns the element with the given index in the given sequence. If the
   -- index is not less than the number of elements of the sequence, then the error
   -- INVALID_INDEX is raised.
```

```
procedure insert (
    list     : in out   elements.sequence;
    item     : in       element;
    index    : in       Pcte.natural := Pcte.natural'LAST;
    status   : in       Pcte_error.handle := EXCEPTION_ONLY);

-- elements.insert inserts the given element in the given sequence immediately before
-- the element with the given index, or if the index is not less than the number of
-- elements of the sequence, the given element is appended after the last element.

procedure replace (
    list     : in out   elements.sequence;
    item     : in       element;
    index    : in       Pcte.natural := Pcte.natural'LAST;
    status   : in       Pcte_error.handle := EXCEPTION_ONLY);

-- elements.replace replaces the element with the given index by the given element, or if
-- the index is not less than the number of elements of the sequence, the given element
-- is appended after the last element.

procedure append (
    list     : in out   elements.sequence;
    item     : in       element;
    status   : in       Pcte_error.handle := EXCEPTION_ONLY);

-- elements.append appends the element to the sequence.

procedure delete (
    list     : in out   elements.sequence;
    index    : in       Pcte.natural := Pcte.natural'FIRST;
    count    : in       Pcte.positive := Pcte.positive'LAST;
    status   : in       Pcte_error.handle := EXCEPTION_ONLY);

-- elements.delete deletes up to the given count elements from the element with the
-- given index from the given sequence.  The number of elements deleted is the lesser
-- of count and the number of elements from the element of list with the given index to
-- the end.

procedure copy (
    into_list     : in out   elements.sequence;
    from_list     : in       elements.sequence;
    into_index    : in       Pcte.natural := Pcte.natural'LAST;
    from_index    : in       Pcte.natural := Pcte.natural'FIRST;
    count         : in       Pcte.positive := Pcte.positive'LAST;
    status        : in       Pcte_error.handle := EXCEPTION_ONLY);

-- elements.copy adds up to the given count elements from the element with index
-- from_index of from_list to into_list.  The elements are inserted immediately before
-- the element of into_list with index into_index, or, if into_index is not less than the
-- number of elements of into_list, are appended to the end of into_list.  The number of
-- elements added is the lesser of count and the number of elements from the element of
-- from_list with index from_index to the end.
```

```
    function length_of (
        list        : elements.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.natural;

    -- elements.length_of returns the number of elements in the given sequence.

    function index_of (
        list        : elements.sequence;
        item        : element;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.integer;

    -- elements.index_of returns the index of the first occurrence of the given element in the
    -- given sequence if the element is a member of the sequence, and -1 otherwise.

    function are_equal (
        first       : elements.sequence;
        second      : elements.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.boolean;

    -- elements.are_equal returns TRUE if the two sequences first and second have the
    -- same number of elements and their corresponding elements are equal, and FALSE
    -- otherwise.

    procedure normalize (
        list    : in out    elements.sequence;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);

    -- elements.normalize reorders the elements of the given sequence in an
    -- implementation-defined canonical order, and deletes any duplicate elements.

    procedure discard (
        list    : in out    elements.sequence;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end elements;
```

The Ada generated datatype sequence declared by the package is limited private. Objects of this type may only be manipulated by the subprograms defined in that package; in particular:

- assignment is not supported;
- all elements of such objects always have defined values.

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| Equal(s1,s2) | are_equal(s1, s2) |
| Head(s) | get(s) |
| IsEmpty(s) | length_of(s) = 0 |
| s:=Tail(s) | delete(s,1,1); |
| s:=Append(s,e) | append(s,e); |
| s:=Empty() | delete(s); |
| Get(s,i) | get(s,i) |
| s:=Put(s,e,i) | insert(s,e,i); |
| s:=Delete(s,i) | delete(s,i,1); |
| s1:=InsertSequence(s1,s2,i) | copy(s1,s2,i); |
| s1:=AppendSequence(s1,s2) | copy(s1,s2); |
| LengthOf(s) | length_of(s) |
| IsMember(s,e) | index_of(s,e) /= 0 |
| s:=Normalize(s) | normalize(s); |

### 8.2.9 LI datatype generator: bounded-set

The LI datatype *bounded-set of base* is mapped to an Ada datatype defined as an array of Pcte.boolean, with an enumeration index type mapping the names of element values to positions in the array. Each LI datatype *bounded set of xxx* is mapped to an Ada datatype called set_of_xxxs (where xxxs is the plural of xxx). An array value of the Ada datatype represents a set including a element value if the corresponding entry in the array has the value TRUE. Thus a bounded set of base type xxx with possible element values VAL1, VAL2, ... VALn is mapped as follows:

**type** xxx **is** (VAL1, VAL2, ... VALn);

**type** xxxs **is array** (xxx) **of** Pcte.boolean;

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| IsIn (s,e) | s (e) |
| Subset (s1,s2) | **for** i **in** xxx **loop**<br>  **if** s1(i) **and not** s2(i) **then**<br>  **return** FALSE;<br>  **end if** ;<br>**end loop** ;<br>**return** TRUE; |
| Equal (s1,s2) | s1 = s2 |
| Difference (s1, s2) | s1 **and not** s2 |
| Union (s1,s2) | s1 **or** s2 |
| Intersection (s1,s2) | s1 **and** s2 |
| Empty () | xxxs'( **others** => FALSE) |
| SetOf (e) | xxxs'(e => TRUE, **others** => FALSE) |
| Select (s) | j : xxx;<br>**for** i **in** xxx **loop**<br>  j := i;<br>  **exit when** s(j);<br>**end loop** ;<br>**return** j; |



### 8.2.10 LI datatype: record

The LI record datatype *xxx = record of (COMPONENT-1: type-1, COMPONENT-2: type-2, ...)* is mapped to the Ada record datatype:

```
type xxx is record
    component_1:    type_1;
    component_2:    type_2;
    component_3:    type_3;
    ...;
end record;
```

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| Equal (x, y) | x = y |
| FieldSelect.component (x) | x.component; |
| Aggregate ( component_1, component_2, ... ) | ( component_1, component_2, ... ) |

### 8.2.11 LI datatype: private

The LI datatype private is mapped in each case to an Ada private type.

### 8.2.12 LI enumerated datatype: pcte-xxx

The LI datatype *pcte-xxx*, defined as *enumerated (val1, val2, ...)*, corresponds to the PCTE enumeration datatype xxx (where the values of xxx are VAL1, VAL2, ...). It is mapped to the Ada datatype xxx, defined as follows.

**type** xxx **is** (VAL1, VAL2, ...);

**Characterizing operations**

| Operation | Ada Operation |
|---|---|
| Equal (x, y) | x = y |
| InOrder (x, y) | x <= y |
| Successor (x) | x'SUCC |

## 8.3 Deriving Ada subprogram semantics from the abstract specification

Each Ada subprogram corresponds to the abstract operation in ISO/IEC 13719-1 whose name precedes the subprogram declaration, as a comment with the form of a clause heading from ISO/IEC 13719-1 e.g.: -- 11.2.9 VOLUME_MOUNT.

In cases where there is not a one-to-one correspondence between abstract operations and Ada subprograms, an explanation is given as a comment before the first Ada subprogram declaration. Where Ada subprogram declarations are out of the order with respect to order of ISO/IEC 13719-1, a cross-reference is given.

The semantics of an Ada subprogram is generally the same as that of the corresponding abstract operation, and is derived as follows in general. Any exceptions are described where they occurs in clause 9 to 22.

- Ada formal parameters of mode **in** correspond to abstract operation parameters of the same names.

- Ada formal parameter types correspond to the PCTE datatypes of the corresponding abstract operation parameters as defined in clause 8.
- Returned values corresponding to results of abstract operations are returned either as function results or as parameters of mode **out** or **in out**. In the latter case, if the result of an operation is a single non-optional value, then the Ada result parameter has the same name.
- Where an abstract operation has optional parameters, and in other cases where the above rules cannot be applied, an explanation is in general given as a comment immediately after the subprogram declaration. There are two general methods of mapping optional parameters and results where no explanation is given with the subprogram. One is where an optional parameter is of a string type and its absence is indicated by providing an empty string value. The other is where an optional parameter or result is dealt with by providing two overloaded subprograms, one with and one without the corresponding parameter.
- In order to use good Ada style, formal parameters with default values (corresponding to optional parameters of the abstract operation) are placed at the end of the parameter list. Otherwise the order of parameters is the same as for the abstract operation.
- In general the names of parameters and results are the same as in the abstract operation. Where that is not possible, because the name in the abstract operation is an Ada reserved word, or in the case of the name 'status' which is used specially in the Ada Binding, the characters 'pcte_' are prefixed to the name in the abstract operation.
- A parameter or result of a union type (excluding enumeration types) is usually handled by providing a set of overloaded subprograms, one for each member of the type union. No explanation is given with the subprogram declarations for such cases.
- Threads in the Abstract Specification are mapped onto tasks in the Ada binding. A single PCTE process executes by the execution of a single Ada program, even when that program includes multiple tasks. Within the process, Ada tasks execute in parallel (proceed independently) in accordance with the rules in ISO 8652 9(5).
- When a task executes a PCTE operation which depends on the occurrence of some event, that task is suspended pending the occurrence of that event. In this case, other tasks may continue to execute and to execute PCTE operations, subject to the Ada tasking rules. The suspended task has no delaying effect on other PCTE operations if the operations to be performed are unrelated.



## 8.4 Package Pcte

```
with Pcte_error, CALENDAR;

package Pcte is

   use Pcte_error

-------------------------------------  PCTE basic types  -------------------------------------

   subtype boolean is STANDARD.BOOLEAN;

   -- Pcte.boolean corresponds to the PCTE datatype Boolean.

   type integer is range implementation-defined;
```

```
-- Pcte.integer corresponds to the PCTE datatype Integer.
type natural is range 0 .. Pcte.integer'LAST;
-- Pcte.natural corresponds to the PCTE datatype Natural.
type float is digits implementation-defined range implementation-defined;
-- Pcte.float corresponds to the PCTE datatype Float.
subtype text is STANDARD.STRING;
-- Pcte.text corresponds to the PCTE datatype Text.
subtype string is STANDARD.STRING;
subtype string_length is STANDARD.NATURAL;
-- Pcte.string corresponds to the PCTE datatype String.
subtype key is Pcte.text;
-- Pcte.key corresponds to the PCTE datatype Key.
subtype actual_key is Pcte.text;
-- Pcte.actual_key corresponds to the PCTE datatype Actual_key.
subtype link_name is Pcte.text;
-- Pcte.link_name corresponds to the PCTE datatype Link_name.
subtype name is Pcte.text;
-- Pcte.name corresponds to the PCTE datatype Name.
subtype type_name is Pcte.text;
-- Pcte.type_name corresponds to the PCTE datatype Type_name.
subtype type_name_in_sds is Pcte.text (1..MAX_NAME_SIZE);
-- Pcte.type_name corresponds to the PCTE datatype Type_name_in_sds.
subtype positive is STANDARD.POSITIVE;
-- Pcte.positive is used to define a character position within Pcte.string or Pcte.text.
package calendar is
    type time is private;
    -- Pcte.calendar.time corresponds to the PCTE datatype Time; it is an Ada private
    -- type defined with its associated subprograms in this package Pcte.calendar. The
    -- subtypes and subprograms have the same meanings as their namesakes in package
    -- CALENDAR.
    DEFAULT_TIME: constant time;
    Pcte.calendar.DEFAULT_TIME is the default value of time attributes.
    subtype year_number is STANDARD.CALENDAR.YEAR_NUMBER
                range 1980 .. 2044;
```

```
subtype month_number is STANDARD.CALENDAR.MONTH_NUMBER;
subtype day_number is STANDARD.CALENDAR.DAY_NUMBER;
subtype day_duration is STANDARD.CALENDAR.DAY_DURATION;
function clock return Pcte.calendar.time;
function year (
    date        : Pcte.calendar.time)
    return      Pcte.calendar.year_number;
function month (
    date        : Pcte.calendar.time)
    return      Pcte.calendar.month_number;
function day  (
    date        : Pcte.calendar.time)
    return      Pcte.calendar.day_number;
function seconds (
    date        : Pcte.calendar.time)
    return      Pcte.calendar.day_duration;
procedure split (
    date        : in    Pcte.calendar.time;
    year        : out   Pcte.calendar.year_number;
    month       : out   Pcte.calendar.month_number;
    day         : out   Pcte.calendar.day_number;
    seconds     : out   Pcte.calendar.day_duration);
function time_of (
    year        : Pcte.calendar.year_number;
    month       : Pcte.calendar.month_number;
    day         : Pcte.calendar.day_number;
    seconds     : Pcte.calendar.day_duration := 0.0)
    return      Pcte.calendar.time;
function "+" (
    left        : Pcte.calendar.time;
    right       : STANDARD.DURATION)
    return      Pcte.calendar.time;
function "+" (
    left        : STANDARD.DURATION;
    right       : Pcte.calendar.time)
    return      Pcte.calendar.time;
function "-" (
    left        : Pcte.calendar.time;
    right       : STANDARD.DURATION)
    return      Pcte.calendar.time;
```

```
        function "-" (
            left        : Pcte.calendar.time;
            right       : Pcte.calendar.time)
            return      STANDARD.DURATION;

        function "<" (
            left        : Pcte.calendar.time;
            right       : Pcte.calendar.time)
            return      Pcte.boolean;

        function "<=" (
            left        : Pcte.calendar.time;
            right       : Pcte.calendar.time)
            return      Pcte.boolean;

        function ">" (
            left        : Pcte.calendar.time;
            right       : Pcte.calendar.time)
            return      Pcte.boolean;

        function ">=" (
            left        : Pcte.calendar.time;
            right       : Pcte.calendar.time)
            return      Pcte.boolean;

        function extend (
            date        : Pcte.calendar.time)
            return      STANDARD.CALENDAR.TIME;

        -- Pcte.calendar.extend converts the value of the parameter date to the type
        -- CALENDAR.TIME.

        function round (
            date        : STANDARD.CALENDAR.TIME)
            return      Pcte.calendar.time;

        -- Pcte.calendar.round converts the value of the parameter date to the type
        -- Pcte.calendar.time.

        TIME_ERROR : exception renames STANDARD.CALENDAR.TIME_ERROR;

    private
        implementation-defined
    end calendar;

-------------------------------        PCTE references        -------------------------------

    package reference is

        type    object_ref      is limited private;

        type    link_ref        is limited private;

        type    type_ref        is limited private;
```

```
subtype attribute_ref is type_ref;

CURRENT_PROCESS          : constant Pcte.reference.object_ref;

LOCAL_WORKSTATION        : constant Pcte.reference.object_ref;

LOCAL_EXECUTION_SITE     : constant Pcte.reference.object_ref;

subtype pathname is Pcte.string;

--  Pcte.reference.pathname corresponds to the PCTE datatype Pathname.

subtype relative_pathname is Pcte.string;

--  Pcte.reference.relative_pathname corresponds to the PCTE datatype
--  Relative_pathname.

type evaluation_point is (NOW, FIRST_USE, EVERY_USE);

--  Pcte.reference.evaluation_point corresponds to the PCTE datatype
--  Evaluation_point.

type evaluation_status is (INTERNAL, EXTERNAL, UNDEFINED);

--      Pcte.reference.evaluation_status corresponds to the PCTE datatype
--      Evaluation_status.  The value UNDEFINED is binding-defined and signifies an
--      object reference which is unset.

type reference_equality is (EQUAL_REFS, UNEQUAL_REFS, EXTERNAL_REFS);

--      Pcte.reference.reference_equality corresponds to the PCTE datatype
--      Reference_equality.

type key_kind is (NATURAL_KEY, STRING_KEY);

type key_value (
    kind          : Pcte.reference.key_kind := NATURAL_KEY;
    string_length : Pcte.string_length := 0)
is record
    case kind is
        when NATURAL_KEY => natural_value : Pcte.natural;
        when STRING_KEY  => string_value  : Pcte.text (1..string_length);
    end case;
end record;

----------------------------------  Object Reference Operations  ----------------------------------

--  23.2.1 OBJECT_REFERENCE_COPY

procedure copy (
    reference     : in  Pcte.reference.object_ref;
    point         : in  Pcte.reference.evaluation_point;
    new_reference : out Pcte.reference.object_ref;
    status        : in  Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 23.2.2 OBJECT_REFERENCE_GET_EVALUATION_POINT

function get_evaluation_point  (
    reference  : Pcte.reference.object_ref;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.reference.evaluation_point;

-- 23.2.3 OBJECT_REFERENCE_GET_PATH

function get_path (
    reference  : Pcte.reference.object_ref;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.reference.pathname;

-- 23.2.4 OBJECT_REFERENCE_GET_STATUS

function get_status (
    reference  : Pcte.reference.object_ref;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.reference.evaluation_status;

-- 23.2.5 OBJECT_REFERENCE_SET_ABSOLUTE

procedure set_absolute (
    pathname        : in    Pcte.reference.pathname;
    point           : in    Pcte.reference.evaluation_point;
    new_reference   : out   Pcte.reference.object_ref;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 23.2.6 OBJECT_REFERENCE_SET_RELATIVE

procedure set_relative (
    reference       : in    Pcte.reference.object_ref;
    pathname        : in    Pcte.reference.relative_pathname;
    point           : in    Pcte.reference.evaluation_point;
    new_reference   : out   Pcte.reference.object_ref;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 23.2.7 OBJECT_REFERENCE_UNSET

procedure unset (
    reference  : in out   Pcte.reference.object_ref;
    status     : in       Pcte_error.handle := EXCEPTION_ONLY);

-- 23.2.8 OBJECT_REFERENCES_ARE_EQUAL

function are_equal (
    first_reference     : Pcte.reference.object_ref;
    second_reference    : Pcte.reference.object_ref;
    status              : Pcte_error.handle := EXCEPTION_ONLY)
    return                Pcte.reference.reference_equality;
```

---------- Link Reference Operations ----------

```
-- 23.3.1 LINK_REFERENCE_COPY

procedure copy (
    reference       : in    Pcte.reference.link_ref;
    point           : in    Pcte.reference.evaluation_point;
    new_reference   : out   Pcte.reference.link_ref;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 23.3.2 LINK_REFERENCE_GET_EVALUATION_POINT

function get_evaluation_point (
    reference  : Pcte.reference.link_ref;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.reference.evaluation_point;

-- 23.3.3 LINK_REFERENCE_GET_KEY

function get_key (
    reference  : Pcte.reference.link_ref;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.key;

-- 23.3.4 LINK_REFERENCE_GET_KEY_VALUE

function get_key_value (
    reference  : Pcte.reference.link_ref;
    index      : Pcte.natural;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.reference.key_value;

-- 23.3.5 LINK_REFERENCE_GET_NAME

function get_name (
    reference  : Pcte.reference.link_ref;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.link_name;

-- 23.3.6 LINK_REFERENCE_GET_STATUS

function get_status (
    reference  : Pcte.reference.link_ref;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.reference.evaluation_status;

-- 23.3.7 LINK_REFERENCE_GET_TYPE

procedure get_type (
    reference       : in    Pcte.reference.link_ref;
    type_reference  : out   Pcte.reference.type_ref;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 23.3.8 LINK_REFERENCE_SET

procedure set (
    link_name       : in    Pcte.link_name;
    point           : in    Pcte.reference.evaluation_point;
    new_reference   : out   Pcte.reference.link_ref;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set (
    link_name       : in    Pcte.reference.type_ref;
    point           : in    Pcte.reference.evaluation_point;
    new_reference   : out   Pcte.reference.link_ref;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set (
    link_name       : in    Pcte.reference.type_ref;
    link_key        : in    Pcte.key;
    point           : in    Pcte.reference.evaluation_point;
    new_reference   : out   Pcte.reference.link_ref;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 23.3.9 LINK_REFERENCE_UNSET

procedure unset (
    reference  : in out   Pcte.reference.link_ref;
    status     : in       Pcte_error.handle := EXCEPTION_ONLY);

-- 23.3.10 LINK_REFERENCES_ARE_EQUAL

function are_equal (
    first_reference     : Pcte.reference.link_ref;
    second_reference    : Pcte.reference.link_ref;
    status              : Pcte_error.handle := EXCEPTION_ONLY)
    return                Pcte.reference.reference_equality;

---------------------------------  Type reference operations  ---------------------------------

-- 23.4.1 TYPE_REFERENCE_COPY

procedure copy (
    reference       : in    Pcte.reference.type_ref;
    point           : in    Pcte.reference.evaluation_point;
    new_reference   : out   Pcte.reference.type_ref;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 23.4.2 TYPE_REFERENCE_GET_EVALUATION_POINT

function get_evaluation_point (
    reference  : Pcte.reference.type_ref;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.reference.evaluation_point;
```

```
    -- 23.4.3 TYPE_REFERENCE_GET_IDENTIFIER

    function get_identifier (
        reference  : Pcte.reference.type_ref;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.type_name;

    -- 23.4.4 TYPE_REFERENCE_GET_NAME

    function get_name (
        reference  : Pcte.reference.type_ref;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.type_name;

    function get_name (
        sds        : Pcte.reference.object_ref;
        reference  : Pcte.reference.type_ref;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.type_name;

    -- 23.4.5 TYPE_REFERENCE_GET_STATUS

    function get_status (
        reference  : Pcte.reference.type_ref;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.reference.evaluation_status;

    -- 23.4.6 TYPE_REFERENCE_SET

    procedure set (
        name           : in     Pcte.type_name;
        point          : in     Pcte.reference.evaluation_point;
        new_reference  : out    Pcte.reference.type_ref;
        status         : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 23.4.7 TYPE_REFERENCE_UNSET

    procedure unset (
        reference  : in out   Pcte.reference.type_ref;
        status     : in       Pcte_error.handle := EXCEPTION_ONLY);

    -- 23.4.8 TYPE_REFERENCES_ARE_EQUAL

    function are_equal (
        first_reference    : Pcte.reference.type_ref;
        second_reference   : Pcte.reference.type_ref;
        status             : Pcte_error.handle := EXCEPTION_ONLY)
        return               Pcte.reference.reference_equality;

private

    implementation-defined
```

```
    end reference;

    subtype object_reference is Pcte.reference.object_ref;
    subtype type_reference is Pcte.reference.type_ref;
    subtype link_reference is Pcte.reference.link_ref;
    subtype attribute_reference is Pcte.reference.attribute_ref;
    -- These types correspond to the PCTE datatypes X_reference.
---------------------------------- PCTE types related to attributes ----------------------------------
    type enumeral_position is range 0 .. Pcte.natural'LAST;
    type value_type is (INTEGER_TYPE, NATURAL_TYPE, BOOLEAN_TYPE,
        TIME_TYPE, FLOAT_TYPE, STRING_TYPE, ENUMERAL_TYPE);
    -- Pcte.value_type corresponds to the PCTE datatype Value_type. ENUMERAL_TYPE
    -- indicates an enumeral type, defined by its position in the corresponding attribute type.
    type attribute_value (
        type_is        : Pcte.value_type := INTEGER_TYPE;
        string_length  : Pcte.string_length := 0)
    is record
        case type_is is
            when BOOLEAN_TYPE =>
                boolean_value    : Pcte.boolean;
            when NATURAL_TYPE =>
                natural_value    : Pcte.natural;
            when INTEGER_TYPE =>
                integer_value    : Pcte.integer;
            when FLOAT_TYPE =>
                float_value      : Pcte.float;
            when TIME_TYPE =>
                time_value       : Pcte.calendar.time;
            when STRING_TYPE =>
                string_value     : Pcte.string(1..string_length);
            when ENUMERAL_TYPE =>
                enumeral_value   : Pcte.enumeral_position;
        end case;
    end record;
    type attribute_assignment (
        type_is        : Pcte.value_type := INTEGER_TYPE;
        string_length  : Pcte.string_length := 0)
    is record
        attribute   : Pcte.type_reference;
        value       : Pcte.attribute_value(type_is, string_length);
    end record;
```

```
    --      Pcte.attribute_assignment corresponds to the PCTE datatype Attribute_assignment.
    --      Pcte.attribute_value is the type of the value field, and is discriminated by the type
    --      of the attribute value.

---------------------------------  PCTE types related to links  ---------------------------------

    type category is (COMPOSITION_LINK, EXISTENCE_LINK, REFERENCE_LINK,
        DESIGNATION_LINK, IMPLICIT_LINK);

    -- Pcte.category corresponds to the PCTE datatype Category.

    type categories is array (Pcte.category) of Pcte.boolean;

    -- Pcte.categories corresponds to the PCTE datatype Categories.

    type object_scope is (ATOMIC, COMPOSITE);

    -- Pcte.object_scope corresponds to the PCTE datatype Object_scope.

---------------------------------  PCTE types related to objects  ---------------------------------

    subtype exact_identifier is Pcte.text;

    -- Pcte.exact_identifier corresponds to the PCTE datatype Exact_identifier.

---------------------------------  sequence packages  ---------------------------------

    -- The semantics of the operations of these packages are defined in 8.2.8.


    package object_references is

        type sequence is limited private;

        -- Pcte.object_references.sequence corresponds to the PCTE datatype Object_references.

        function get (
            list      : Pcte.object_references.sequence;
            index     : Pcte.natural := Pcte.natural'FIRST;
            status    : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.object_reference;

        procedure insert (
            list      : in out   Pcte.object_references.sequence;
            item      : in       Pcte.object_reference;
            index     : in       Pcte.natural := Pcte.natural'LAST;
            status    : in       Pcte_error.handle := EXCEPTION_ONLY);

        procedure replace (
            list      : in out   Pcte.object_references.sequence;
            item      : in       Pcte.object_reference;
            index     : in       Pcte.natural := Pcte.natural'LAST;
            status    : in       Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure append (
        list        : in out    Pcte.object_references.sequence;
        item        : in        Pcte.object_reference;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure delete (
        list        : in out    Pcte.object_references.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure copy (
        into_list       : in out    Pcte.object_references.sequence;
        from_list       : in        Pcte.object_references.sequence;
        into_index      : in        Pcte.natural := Pcte.natural'LAST;
        from_index      : in        Pcte.natural := Pcte.natural'FIRST;
        count           : in        Pcte.positive := Pcte.positive'LAST;
        status          : in        Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list        : Pcte.object_references.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.natural;

    function index_of (
        list        : Pcte.object_references.sequence;
        item        : Pcte.object_reference;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.integer;

    function are_equal (
        first       : Pcte.object_references.sequence;
        second      : Pcte.object_references.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte.object_references.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte.object_references.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end object_references;


package link_references is

    type sequence is limited private;
```

```
-- Pcte.link_references.sequence corresponds to the PCTE datatype Link_references.

function get (
    list        : Pcte.link_references.sequence;
    index       : Pcte.natural := Pcte.natural'FIRST;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.link_reference;

procedure insert (
    list        : in out    Pcte.link_references.sequence;
    item        : in        Pcte.link_reference;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure replace (
    list        : in out    Pcte.link_references.sequence;
    item        : in        Pcte.link_reference;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure append (
    list        : in out    Pcte.link_references.sequence;
    item        : in        Pcte.link_reference;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure delete (
    list        : in out    Pcte.link_references.sequence;
    index       : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure copy (
    into_list   : in out    Pcte.link_references.sequence;
    from_list   : in        Pcte.link_references.sequence;
    into_index  : in        Pcte.natural := Pcte.natural'LAST;
    from_index  : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

function length_of (
    list        : Pcte.link_references.sequence;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.natural;

function index_of (
    list        : Pcte.link_references.sequence;
    item        : Pcte.link_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.integer;
```

```
    function are_equal (
        first       : Pcte.link_references.sequence;
        second      : Pcte.link_references.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte.link_references.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte.link_references.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end link_references;


package type_references is

    type sequence is limited private;

    --  Pcte.type_references.sequence corresponds to the PCTE datatype Type_references.

    function get (
        list        : Pcte.type_references.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.type_reference;

    procedure insert (
        list        : in out    Pcte.type_references.sequence;
        item        : in        Pcte.type_reference;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure replace (
        list        : in out    Pcte.type_references.sequence;
        item        : in        Pcte.type_reference;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure append (
        list        : in out    Pcte.type_references.sequence;
        item        : in        Pcte.type_reference;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure delete (
        list        : in out    Pcte.type_references.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure copy (
        into_list      : in out   Pcte.type_references.sequence;
        from_list      : in       Pcte.type_references.sequence;
        into_index     : in       Pcte.natural := Pcte.natural'LAST;
        from_index     : in       Pcte.natural := Pcte.natural'FIRST;
        count          : in       Pcte.positive := Pcte.positive'LAST;
        status         : in       Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list       : Pcte.type_references.sequence;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.natural;

    function index_of (
        list       : Pcte.type_references.sequence;
        item       : Pcte.type_reference;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.integer;

    function are_equal (
        first      : Pcte.type_references.sequence;
        second     : Pcte.type_references.sequence;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.boolean;

    procedure normalize (
        list       : in out   Pcte.type_references.sequence;
        status     : in       Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list       : in out   Pcte.type_references.sequence;
        status     : in       Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end type_references;


package type_names_in_sds is

    type sequence is limited private;

    -- Pcte.type_names_in_sds.sequence corresponds to the PCTE datatype
    -- Type_names_in_sds.

    function get (
        list       : Pcte.type_names_in_sds.sequence;
        index      : Pcte.natural := Pcte.natural'FIRST;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.type_name_in_sds;
```

```
procedure insert (
    list        : in out    Pcte.type_names_in_sds.sequence;
    item        : in        Pcte.type_name_in_sds;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure replace (
    list        : in out    Pcte.type_names_in_sds.sequence;
    item        : in        Pcte.type_name_in_sds;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure append (
    list        : in out    Pcte.type_names_in_sds.sequence;
    item        : in        Pcte.type_name_in_sds;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure delete (
    list        : in out    Pcte.type_names_in_sds.sequence;
    index       : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure copy (
    into_list       : in out    Pcte.type_names_in_sds.sequence;
    from_list       : in        Pcte.type_names_in_sds.sequence;
    into_index      : in        Pcte.natural := Pcte.natural'LAST;
    from_index      : in        Pcte.natural := Pcte.natural'FIRST;
    count           : in        Pcte.positive := Pcte.positive'LAST;
    status          : in        Pcte_error.handle := EXCEPTION_ONLY);

function length_of (
    list        : Pcte.type_names_in_sds.sequence;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.natural;

function index_of (
    list        : Pcte.type_names_in_sds.sequence;
    item        : Pcte.type_name_in_sds;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.integer;

function index_of (
    list        : Pcte.type_names_in_sds.sequence;
    item        : Pcte.type_name_in_sds;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.integer;

procedure normalize (
    list        : in out    Pcte.type_names_in_sds.sequence;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure discard (
        list        : in out    Pcte.type_names_in_sds.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
private
    implementation-defined
end type_names_in_sds;


package attribute_references is
    type sequence is limited private;
    -- Pcte.attribute_references.sequence corresponds to the PCTE datatype
    -- Attribute_references.
    function get (
        list        : Pcte.attribute_references.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.attribute_reference;
    procedure insert (
        list        : in out    Pcte.attribute_references.sequence;
        item        : in        Pcte.attribute_reference;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
    procedure replace (
        list        : in out    Pcte.attribute_references.sequence;
        item        : in        Pcte.attribute_reference;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
    procedure append (
        list        : in out    Pcte.attribute_references.sequence;
        item        : in        Pcte.attribute_reference;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
    procedure delete (
        list        : in out    Pcte.attribute_references.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
    procedure copy (
        into_list   : in out    Pcte.attribute_references.sequence;
        from_list   : in        Pcte.attribute_references.sequence;
        into_index  : in        Pcte.natural := Pcte.natural'LAST;
        from_index  : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    function length_of (
        list        : Pcte.attribute_references.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.natural;

    function index_of (
        list        : Pcte.attribute_references.sequence;
        item        : Pcte.attribute_reference;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.integer;

    function are_equal (
        first       : Pcte.attribute_references.sequence;
        second      : Pcte.attribute_references.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte.attribute_references.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte.attribute_references.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end attribute_references;


package attribute_assignments is

    type sequence is limited private;

    -- Pcte.attribute_assignments.sequence corresponds to the PCTE datatype
    -- Attribute_assignments.

    function get (
        list        : Pcte.attribute_assignments.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.attribute_assignment;

    procedure insert (
        list        : in out    Pcte.attribute_assignments.sequence;
        item        : in        Pcte.attribute_assignment;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure replace (
        list        : in out    Pcte.attribute_assignments.sequence;
        item        : in        Pcte.attribute_assignment;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure append (
        list        : in out    Pcte.attribute_assignments.sequence;
        item        : in        Pcte.attribute_assignment;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure delete (
        list        : in out    Pcte.attribute_assignments.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure copy (
        into_list   : in out    Pcte.attribute_assignments.sequence;
        from_list   : in        Pcte.attribute_assignments.sequence;
        into_index  : in        Pcte.natural := Pcte.natural'LAST;
        from_index  : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list        : Pcte.attribute_assignments.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.natural;

    function index_of (
        list        : Pcte.attribute_assignments.sequence;
        item        : Pcte.attribute_assignment;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.integer;

    function are_equal (
        first       : Pcte.attribute_assignments.sequence;
        second      : Pcte.attribute_assignments.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte.attribute_assignments.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte.attribute_assignments.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined
```

```
    end attribute_assignments;
end Pcte;
```

# 9 Object managment

## 9.1 Object management datatypes

```
-- See the beginning of the packages Pcte_link, Pcte_object and Pcte_version.
```

## 9.2 Link operations

```
with Pcte, Pcte_error, Pcte_archive, Pcte_volume;
package Pcte_link is
    use Pcte, Pcte.calendar, Pcte_error;

    -- 9.2.1 LINK_CREATE

    procedure create (
        origin      : in    Pcte.object_reference;
        new_link    : in    Pcte.link_reference;
        dest        : in    Pcte.object_reference;
        reverse_key : in    Pcte.actual_key := "";
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 9.2.2 LINK_DELETE

    procedure delete (
        origin : in    Pcte.object_reference;
        link   : in    Pcte.link_reference;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 9.2.3 LINK_DELETE_ATTRIBUTE

    procedure delete_attribute (
        origin    : in    Pcte.object_reference;
        link      : in    Pcte.link_reference;
        attribute : in    Pcte.attribute_reference;
        status    : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 9.2.4 LINK_GET_ATTRIBUTE

    -- LINK_GET_ATTRIBUTE is mapped to overloaded versions of the function
    -- Pcte_link.get_attribute, one for each possible type of the result. If the type of
    -- the result of the particular overloaded function that is used is different from the
    -- type of its parameter attribute, then the error VALUE_TYPE_IS_INVALID is
    -- raised.
```

```
function get_attribute (
    origin      : Pcte.object_reference;
    link        : Pcte.link_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.boolean;

function get_attribute (
    origin      : Pcte.object_reference;
    link        : Pcte.link_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.integer;

function get_attribute (
    origin      : Pcte.object_reference;
    link        : Pcte.link_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.natural;

function get_attribute (
    origin      : Pcte.object_reference;
    link        : Pcte.link_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.float;

function get_attribute (
    origin      : Pcte.object_reference;
    link        : Pcte.link_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.calendar.time;

function get_attribute (
    origin      : Pcte.object_reference;
    link        : Pcte.link_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.enumeral_position;

function get_attribute (
    origin      : Pcte.object_reference;
    link        : Pcte.link_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.string;
```

```
-- 9.2.5 LINK_GET_DESTINATION_VOLUME

function get_destination_volume (
    origin      : Pcte.object_reference;
    link        : Pcte.link_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte_volume.volume_info;

-- 9.2.6 LINK_GET_KEY

function get_key (
    origin      : Pcte.object_reference;
    link        : Pcte.link_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.actual_key;

-- 9.2.7 LINK_GET_REVERSE

procedure get_reverse (
    origin          : in       Pcte.object_reference;
    link            : in       Pcte.link_reference;
    reverse_link    : in out   Pcte.link_reference;
    dest            : in out   Pcte.object_reference;
    status          : in       Pcte_error.handle := EXCEPTION_ONLY);

-- 9.2.8 LINK_GET_SEVERAL_ATTRIBUTES

--  The effect of assigning VISIBLE_ATTRIBUTE_TYPES to the parameter attributes is
--  achieved by the first overloaded subprogram; the effect of assigning
--  Attribute_designators to the parameter attributes is achieved by the second overloaded
--  subprogram.

procedure get_several_attributes (
    origin      : in       Pcte.object_reference;
    link        : in       Pcte.link_reference;
    values      : in out   Pcte.attribute_assignments.sequence;
    status      : in       Pcte_error.handle := EXCEPTION_ONLY);

procedure get_several_attributes (
    origin      : in       Pcte.object_reference;
    link        : in       Pcte.link_reference;
    attributes  : in       Pcte.attribute_references.sequence;
    values      : in out   Pcte.attribute_assignments.sequence;
    status      : in       Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 9.2.9 LINK_REPLACE

procedure replace (
    origin          : in    Pcte.object_reference;
    link            : in    Pcte.link_reference;
    new_origin      : in    Pcte.object_reference;
    new_link        : in    Pcte.link_reference;
    new_reverse_key : in    Pcte.actual_key := "";
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 9.2.10 LINK_RESET_ATTRIBUTE

procedure reset_attribute (
    origin      : in    Pcte.object_reference;
    link        : in    Pcte.link_reference;
    attribute   : in    Pcte.attribute_reference;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 9.2.11 LINK_SET_ATTRIBUTE

-- LINK_SET_ATTRIBUTE is mapped to overloaded versions of the procedure
-- Pcte_link.set_attribute, one for each possible type of the parameter value.  If the type
-- of the parameter value of the particular overloaded procedure that is used is different
-- from the type of its parameter attribute, then the error VALUE_TYPE_IS_INVALID is
-- raised.

procedure set_attribute (
    origin      : in    Pcte.object_reference;
    link        : in    Pcte.link_reference;
    attribute   : in    Pcte.attribute_reference;
    value       : in    Pcte.boolean := FALSE;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set_attribute (
    origin      : in    Pcte.object_reference;
    link        : in    Pcte.link_reference;
    attribute   : in    Pcte.attribute_reference;
    value       : in    Pcte.integer := 0;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set_attribute (
    origin      : in    Pcte.object_reference;
    link        : in    Pcte.link_reference;
    attribute   : in    Pcte.attribute_reference;
    value       : in    Pcte.natural := 0;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure set_attribute (
        origin      : in    Pcte.object_reference;
        link        : in    Pcte.link_reference;
        attribute   : in    Pcte.attribute_reference;
        value       : in    Pcte.float := 0.0;
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);

    procedure set_attribute (
        origin      : in    Pcte.object_reference;
        link        : in    Pcte.link_reference;
        attribute   : in    Pcte.attribute_reference;
        value       : in    Pcte.string := "";
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);

    procedure set_attribute (
        origin      : in    Pcte.object_reference;
        link        : in    Pcte.link_reference;
        attribute   : in    Pcte.attribute_reference;
        value       : in    Pcte.calendar.time := DEFAULT_TIME;
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);

    procedure set_attribute (
        origin      : in    Pcte.object_reference;
        link        : in    Pcte.link_reference;
        attribute   : in    Pcte.attribute_reference;
        value       : in    Pcte.enumeral_position;
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 9.2.12 LINK_SET_SEVERAL_ATTRIBUTES

    procedure set_several_attributes (
        origin      : in    Pcte.object_reference;
        link        : in    Pcte.link_reference;
        attributes  : in    Pcte.attribute_assignments.sequence;
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 11.2.7 LINK_GET_DESTINATION_ARCHIVE

    function get_destination_archive (
        origin      : Pcte.object_reference;
        link        : Pcte.link_reference;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte_archive.archive_identifier;

end Pcte_link;
```

## 9.3 Object operations

```
with Pcte, Pcte_error, Pcte_discretionary, Pcte_volume;

package Pcte_object is
```

```
use Pcte, Pcte.calendar, Pcte_error;

type type_ancestry is (EQUAL_TYPE, ANCESTOR_TYPE, DESCENDANT_TYPE,
    UNRELATED_TYPE);

-- Pcte_object.type_ancestry corresponds to the PCTE datatype Type_ancestry.

type link_scope is (INTERNAL_LINKS, EXTERNAL_LINKS, ALL_LINKS);

-- Pcte_object.link_scope corresponds to the PCTE datatype Link_scope.

type link_set_descriptor is record
    origin  : Pcte.object_reference;
    links   : Pcte.link_references.sequence;
end record;

-- Pcte_object.link_set_descriptor corresponds to the PCTE datatype Link_set_descriptor.


-- The semantics of the operations of this package are defined in 8.2.8.

package link_set_descriptors is

    type sequence is limited private;

    -- Pcte_object.link_set_descriptor.sequence corresponds to the PCTE datatype
    -- Link_set_descriptors.

    function get (
        list        : Pcte_object.link_set_descriptors.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte_object.link_set_descriptor;

    procedure insert (
        list        : in out    Pcte_object.link_set_descriptors.sequence;
        item        : in        Pcte_object.link_set_descriptor;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure replace (
        list        : in out    Pcte_object.link_set_descriptors.sequence;
        item        : in        Pcte_object.link_set_descriptor;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure append (
        list        : in out    Pcte_object.link_set_descriptors.sequence;
        item        : in        Pcte_object.link_set_descriptor;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure delete (
        list        : in out    Pcte_object.link_set_descriptors.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure copy (
        into_list       : in out    Pcte_object.link_set_descriptors.sequence;
        from_list       : in        Pcte_object.link_set_descriptors.sequence;
        into_index      : in        Pcte.natural := Pcte.natural'LAST;
        from_index      : in        Pcte.natural := Pcte.natural'FIRST;
        count           : in        Pcte.positive := Pcte.positive'LAST;
        status          : in        Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list        : Pcte_object.link_set_descriptors.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.natural;

    function index_of (
        list        : Pcte_object.link_set_descriptors.sequence;
        item        : Pcte_object.link_set_descriptor;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.integer;

    function are_equal (
        first       : Pcte_object.link_set_descriptors.sequence;
        second      : Pcte_object.link_set_descriptors.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte_object.link_set_descriptors.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte_object.link_set_descriptors.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

private
    implementation-defined
end link_set_descriptors;


type time_kind is (CURRENT_SYSTEM_TIME, SPECIFIED_TIME);
```

```
type time_selection (kind : Pcte_object.time_kind := CURRENT_SYSTEM_TIME)
is record
    case kind is
        when CURRENT_SYSTEM_TIME =>  null;
        when SPECIFIED_TIME =>       time : Pcte.calendar.time;
    end case;
end record;

-- 9.3.1 OBJECT_CHECK_TYPE

function check_type (
    object      : Pcte.object_reference;
    type2       : Pcte.type_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte_object.type_ancestry;

-- 9.3.2 OBJECT_CONVERT

procedure convert (
    object      : in    Pcte.object_reference;
    pcte_type   : in    Pcte.type_reference;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.3 OBJECT_COPY

procedure copy (
    object          : in        Pcte.object_reference;
    new_origin      : in        Pcte.object_reference;
    new_link        : in        Pcte.link_reference;
    access_mask     : in        Pcte_discretionary.object.atomic_access_rights;
    new_object      : in out    Pcte.object_reference;
    reverse_key     : in        Pcte.actual_key := "";
    status          : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure copy (
    object                  : in        Pcte.object_reference;
    new_origin              : in        Pcte.object_reference;
    new_link                : in        Pcte.link_reference;
    on_same_volume_as       : in        Pcte.object_reference;
    access_mask             : in        Pcte_discretionary.object.atomic_access_rights;
    new_object              : in out    Pcte.object_reference;
    reverse_key             : in        Pcte.actual_key := "";
    status                  : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 9.3.4 OBJECT_CREATE

procedure create (
    pcte_type       : in        Pcte.type_reference;
    new_origin      : in        Pcte.object_reference;
    new_link        : in        Pcte.link_reference;
    access_mask     : in        Pcte_discretionary.object.atomic_access_rights;
    new_object      : in out    Pcte.object_reference;
    reverse_key     : in        Pcte.actual_key := "";
    status          : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure create (
    pcte_type                   : in        Pcte.type_reference;
    new_origin                  : in        Pcte.object_reference;
    new_link                    : in        Pcte.link_reference;
    on_same_volume_as           : in        Pcte.object_reference;
    access_mask                 : in        Pcte_discretionary.object.atomic_access_rights;
    new_object                  : in out    Pcte.object_reference;
    reverse_key                 : in        Pcte.actual_key := "";
    status                      : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.5 OBJECT_DELETE

procedure delete (
    origin  : in    Pcte.object_reference;
    link    : in    Pcte.link_reference;
    status  : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.6 OBJECT_DELETE_ATTRIBUTE

procedure delete_attribute (
    object      : in    Pcte.object_reference;
    attribute   : in    Pcte.attribute_reference;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.7 OBJECT_GET_ATTRIBUTE

-- OBJECT_GET_ATTRIBUTE is mapped to overloaded versions of the function
-- Pcte_object.get_attribute, one for each possible type of the result. If the type of the
-- result of the particular overloaded function that is used is different from the type of its
-- parameter attribute, then the error VALUE_TYPE_IS_INVALID is raised.

function get_attribute (
    object      : Pcte.object_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.boolean;
```

```
function get_attribute (
    object      : Pcte.object_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.natural;

function get_attribute (
    object      : Pcte.object_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.integer;

function get_attribute (
    object      : Pcte.object_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.float;

function get_attribute (
    object      : Pcte.object_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.calendar.time;

function get_attribute (
    object      : Pcte.object_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.string;

function get_attribute (
    object      : Pcte.object_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.enumeral_position;

-- 9.3.8 OBJECT_GET_PREFERENCE

procedure get_preference (
    object          : in        Pcte.object_reference;
    key             : out       Pcte.text;
    key_length      : out       Pcte.string_length;
    pcte_type       : in out    Pcte.type_reference;
    status          : in        Pcte_error.handle := EXCEPTION_ONLY);

-- The key_length parameter gives the actual length of the value returned in the key
-- parameter. The error STRING_IS_TOO_SHORT is returned when the parameter key
-- is too short to hold the returned value.
```

```
-- 9.3.9 OBJECT_GET_SEVERAL_ATTRIBUTES

-- The effect of assigning VISIBLE_ATTRIBUTE_TYPES to the parameter attributes
-- is achieved by the first overloaded subprogram; the effect of assigning
-- Attribute_type_nominators to the parameter attributes is achieved by the second
-- overloaded subprogram.

procedure get_several_attributes (
    object : in      Pcte.object_reference;
    values : in out  Pcte.attribute_assignments.sequence;
    status : in      Pcte_error.handle := EXCEPTION_ONLY);

procedure get_several_attributes (
    object     : in      Pcte.object_reference;
    attributes : in      Pcte.attribute_references.sequence;
    values     : in out  Pcte.attribute_assignments.sequence;
    status     : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.10 OBJECT_GET_TYPE

function get_type (
    object     : Pcte.object_reference;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.type_reference;

-- 9.3.11 OBJECT_IS_COMPONENT

function is_component (
    object1    : Pcte.object_reference;
    object2    : Pcte.object_reference;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.boolean;

-- 9.3.12 OBJECT_LIST_LINKS

-- The effect of assigning ALL_LINK_TYPES to the parameter visibility is achieved by the
-- subprogram Pcte_object.list_all_links.  The effect of assigning VISIBLE_LINK_TYPES
-- to the parameter visibility is achieved by the subprogram Pcte_object.
-- list_links_in_working_schema.  The effect of assigning Link_type_nominators to the
-- parameter visibility is achieved by the subprogram Pcte_object.list_links_of_types.

procedure list_links_in_working_schema (
    origin     : in      Pcte.object_reference;
    extent     : in      Pcte_object.link_scope;
    scope      : in      Pcte.object_scope;
    categories : in      Pcte.categories;
    links      : in out  Pcte_object.link_set_descriptors.sequence;
    status     : in      Pcte_error.handle := EXCEPTION_ONLY);
```

```
procedure list_links_of_types (
    origin      : in      Pcte.object_reference;
    extent      : in      Pcte_object.link_scope;
    scope       : in      Pcte.object_scope;
    categories  : in      Pcte.categories;
    link_types  : in      Pcte.type_references.sequence;
    links       : in out  Pcte_object.link_set_descriptors.sequence;
    status      : in      Pcte_error.handle := EXCEPTION_ONLY);

procedure list_all_links (
    origin      : in      Pcte.object_reference;
    extent      : in      Pcte_object.link_scope;
    scope       : in      Pcte.object_scope;
    categories  : in      Pcte.categories;
    links       : in out  Pcte_object.link_set_descriptors.sequence;
    status      : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.13 OBJECT_LIST_VOLUMES

procedure list_volumes (
    object      : in      Pcte.object_reference;
    volumes     : in out  Pcte_volume.volume_infos.sequence;
    status      : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.14 OBJECT_MOVE

procedure move (
    object                  : in  Pcte.object_reference;
    on_same_volume_as       : in  Pcte.object_reference;
    scope                   : in  Pcte.object_scope;
    status                  : in  Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.15 OBJECT_RESET_ATTRIBUTE

procedure reset_attribute (
    object      : in  Pcte.object_reference;
    attribute   : in  Pcte.attribute_reference;
    status      : in  Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.16 OBJECT_SET_ATTRIBUTE

-- OBJECT_SET_ATTRIBUTE is mapped to overloaded versions of the procedure
-- Pcte_object.set_attribute, one for each possible type of the parameter value. If the type
-- of the parameter value of the particular overloaded procedure that is used is different
-- from the type of its parameter attribute, then the error VALUE_TYPE_IS_INVALID is
-- raised.

procedure set_attribute (
    object      : in  Pcte.object_reference;
    attribute   : in  Pcte.attribute_reference;
    value       : in  Pcte.boolean := FALSE;
    status      : in  Pcte_error.handle := EXCEPTION_ONLY);
```

```
procedure set_attribute (
    object      : in    Pcte.object_reference;
    attribute   : in    Pcte.attribute_reference;
    value       : in    Pcte.natural := 0;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set_attribute (
    object      : in    Pcte.object_reference;
    attribute   : in    Pcte.attribute_reference;
    value       : in    Pcte.integer := 0;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set_attribute (
    object      : in    Pcte.object_reference;
    attribute   : in    Pcte.attribute_reference;
    value       : in    Pcte.float := 0.0;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set_attribute (
    object      : in    Pcte.object_reference;
    attribute   : in    Pcte.attribute_reference;
    value       : in    Pcte.calendar.time := DEFAULT_TIME;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set_attribute (
    object      : in    Pcte.object_reference;
    attribute   : in    Pcte.attribute_reference;
    value       : in    Pcte.string := "";
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set_attribute (
    object      : in    Pcte.object_reference;
    attribute   : in    Pcte.attribute_reference;
    value       : in    Pcte.enumeral_position;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 9.3.17 OBJECT_SET_PREFERENCE

-- The effect of providing both the optional parameter type and the optional parameter key is
-- obtained by the first subprogram. The effect of providing the optional parameter type and
-- not providing the optional parameter key is obtained by the second subprogram. The
-- effect of not providing the optional parameter type and providing the optional parameter
-- key is obtained by the third subprogram. The effect of providing neither the optional
-- parameter type nor the optional parameter key is obtained by the procedure
-- Pcte_object.unset_preference.

procedure set_preference (
    object      : in    Pcte.object_reference;
    pcte_type   : in    Pcte.type_reference;
    key         : in    Pcte.text;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
        procedure set_type_preference (
            object      : in    Pcte.object_reference;
            pcte_type   : in    Pcte.type_reference;
            status      : in    Pcte_error.handle := EXCEPTION_ONLY);

        procedure set_key_preference (
            object  : in    Pcte.object_reference;
            key     : in    Pcte.string;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);

        procedure unset_preference (
            object  : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 9.3.18 OBJECT_SET_SEVERAL_ATTRIBUTES

        procedure set_several_attributes (
            object      : in    Pcte.object_reference;
            attributes  : in    Pcte.attribute_assignments.sequence;
            status      : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 9.3.19 OBJECT_SET_TIME_ATTRIBUTES

        procedure set_time_attributes (
            object              : in    Pcte.object_reference;
            scope               : in    Pcte.object_scope;
            last_access         : in    Pcte_object.time_selection :=
                                            (KIND => CURRENT_SYSTEM_TIME);
            last_modification   : in    Pcte_object.time_selection :=
                                            (KIND => CURRENT_SYSTEM_TIME);
            status              : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 9.3.20 VOLUME_LIST_OBJECTS

        --  See 11.2.

    end     Pcte_object;
```

## 9.4 Version operations

```
    with Pcte, Pcte_error, Pcte_discretionary;

    package Pcte_version is

        use Pcte, Pcte_error;

        type version_relation is (ANCESTOR_VSN, DESCENDANT_VSN, SAME_VSN,
            RELATED_VSN, UNRELATED_VSN);

        --  Pcte_version.version_relation corresponds to the PCTE datatype Version_relation.
```

```
-- 9.4.1 VERSION_ADD_PREDECESSOR

procedure add_predecessor (
    version             : in    Pcte.object_reference;
    new_predecessor     : in    Pcte.object_reference;
    status              : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 9.4.2 VERSION_IS_CHANGED

function is_changed (
    version         : Pcte.object_reference;
    predecessor     : Pcte.actual_key;
    status          : Pcte_error.handle := EXCEPTION_ONLY)
    return            Pcte.boolean;

-- 9.4.3 VERSION_REMOVE

procedure remove (
    version     : in    Pcte.object_reference;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 9.4.4 VERSION_REMOVE_PREDECESSOR

procedure remove_predecessor (
    version         : in    Pcte.object_reference;
    predecessor     : in    Pcte.object_reference;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 9.4.5 VERSION_REVISE

procedure revise (
    version                 : in        Pcte.object_reference;
    new_origin              : in        Pcte.object_reference;
    new_link                : in        Pcte.link_reference;
    on_same_volume_as       : in        Pcte.object_reference;
    access_mask             : in        Pcte_discretionary.object.atomic_access_rights;
    new_version             : in out    Pcte.object_reference;
    status                  : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure revise (
    version         : in        Pcte.object_reference;
    new_origin      : in        Pcte.object_reference;
    new_link        : in        Pcte.link_reference;
    access_mask     : in        Pcte_discretionary.object.atomic_access_rights;
    new_version     : in out    Pcte.object_reference;
    status          : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 9.4.6 VERSION_SNAPSHOT

--  The parameter new_link_and_origin is mapped as two parameters new_origin and
--  new_link for consistency with Pcte_version.revise.
```

```
    procedure snapshot (
        version              : in      Pcte.object_reference;
        new_origin           : in      Pcte.object_reference;
        new_link             : in      Pcte.link_reference;
        on_same_volume_as    : in      Pcte.object_reference;
        access_mask          : in      Pcte_discretionary.object.atomic_access_rights;
        new_version          : in out  Pcte.object_reference;
        status               : in      Pcte_error.handle := EXCEPTION_ONLY);

    procedure snapshot (
        version          : in      Pcte.object_reference;
        new_origin       : in      Pcte.object_reference;
        new_link         : in      Pcte.link_reference;
        access_mask      : in      Pcte_discretionary.object.atomic_access_rights;
        new_version      : in out  Pcte.object_reference;
        status           : in      Pcte_error.handle := EXCEPTION_ONLY);

    procedure snapshot (
        version              : in      Pcte.object_reference;
        on_same_volume_as    : in      Pcte.object_reference;
        access_mask          : in      Pcte_discretionary.object.atomic_access_rights;
        new_version          : in out  Pcte.object_reference;
        status               : in      Pcte_error.handle := EXCEPTION_ONLY);

    procedure snapshot (
        version          : in      Pcte.object_reference;
        access_mask      : in      Pcte_discretionary.object.atomic_access_rights;
        new_version      : in out  Pcte.object_reference;
        status           : in      Pcte_error.handle := EXCEPTION_ONLY);

    -- 9.4.7 VERSION_TEST_ANCESTRY

    function test_ancestry (
        version1  : Pcte.object_reference;
        version2  : Pcte.object_reference;
        status    : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte_version.version_relation;

    -- 9.4.8 VERSION_TEST_DESCENT

    function test_descent (
        version1  : Pcte.object_reference;
        version2  : Pcte.object_reference;
        status    : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte_version.version_relation;

end  Pcte_version;
```

## 10 Schema management

```
with Pcte, Pcte_error;
```

```
package Pcte_sds is

    use Pcte, Pcte_error, Pcte.calendar, Pcte.reference;
```

## 10.1 Schema management datatypes

```
    type duplication is (DUPLICATED, NON_DUPLICATED);

    -- Pcte_sds.duplication corresponds to the PCTE datatype Duplication.

    type exclusiveness is (SHARABLE, EXCLUSIVE);

    -- Pcte_sds.exclusiveness corresponds to the PCTE datatype Exclusiveness.

    type stability is (ATOMIC_STABLE, COMPOSITE_STABLE, NON_STABLE);

    -- Pcte_sds.stability corresponds to the PCTE datatype Stability.

    type contents_type is (NO_CONTENTS, FILE_TYPE, PIPE_TYPE, DEVICE_TYPE,
        AUDIT_FILE_TYPE, ACCOUNTING_LOG_TYPE);

    -- Pcte_sds.contents_type corresponds to the PCTE datatype Contents_type. The value
    -- NO_CONTENTS is used to indicate absence of a Contents_type value.

    type link_type_properties is record
        category       : Pcte.category;
        lower_bound    : Pcte.natural;
        upper_bound    : Pcte.natural;
        exclusiveness  : Pcte_sds.exclusiveness;
        stability      : Pcte_sds.stability;
        duplication    : Pcte_sds.duplication;
        key_types      : Pcte.type_names_in_sds.sequence;
    end record;

    -- Pcte_sds.link_type_properties corresponds to the PCTE datatypes of certain parameters
    -- of SDS_CREATE_RELATIONSHIP_TYPE and SDS_GET_LINK_TYPE_
    -- PROPERTIES, which are mapped to a single parameter.

    type attribute_type_properties (
        type_is        : Pcte.value_type := INTEGER_TYPE;
        string_length  : Pcte.string_length := 0)
    is record
        duplication    : Pcte_sds.duplication;
        value          : Pcte.attribute_value(type_is, string_length);
        enumeral_types : Pcte.type_names_in_sds.sequence
    end record;

    -- Pcte_sds.attribute_type_properties corresponds to the PCTE datatypes of the results of
    -- SDS_GET_ATTRIBUTE_TYPE_PROPERTIES.

    type object_type_properties is record
        contents_type  : Pcte_sds.contents_type;
        parents        : Pcte.type_names_in_sds.sequence;
        children       : Pcte.type_names_in_sds.sequence;
    end record;
```

```
-- Pcte_sds.object_type_properties corresponds to the PCTE datatypes of the results of
-- SDS_GET_OBJECT_TYPE_PROPERTIES.

type definition_mode_value is (CREATE, DELETE, READ, WRITE, NAVIGATE);

type definition_mode_values is array (definition_mode_value) of Pcte.boolean;

-- Pcte_sds.definition_mode_values corresponds to the PCTE datatype
-- Definition_mode_values.

type attribute_scan_kind is (OBJECT, OBJECT_ALL, LINK_KEY, LINK_NON_KEY);

-- Pcte_sds.attribute_scan_kind corresponds to the PCTE datatype Attribute_scan_kind.

type link_scan_kind is (ORIGIN, ORIGIN_ALL, DESTINATION, DESTINATION_ALL,
    KEY, NON_KEY);

-- Pcte_sds.link_scan_kind corresponds to the PCTE datatype Link_scan_kind.

type object_scan_kind is (CHILD, DESCENDANT, PARENT, ANCESTOR,
    ATTRIBUTE, ATTRIBUTE_ALL, LINK_ORIGIN, LINK_ORIGIN_ALL,
    LINK_DESTINATION, LINK_DESTINATION_ALL);

-- Pcte_sds.object_scan_kind corresponds to the PCTE datatype Object_scan_kind.

type type_kind is (OBJECT_TYPE, LINK_TYPE, ATTRIBUTE_TYPE,
    ENUMERAL_TYPE);

-- Pcte_sds.type_kind corresponds to the PCTE datatype Type_kind.
```

## 10.2 Update operations

```
-- 10.2.1 SDS_ADD_DESTINATION

procedure add_destination (
    sds             : in    Pcte.object_reference;
    link_type       : in    Pcte.type_name_in_sds;
    object_type     : in    Pcte.type_name_in_sds;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.2 SDS_APPLY_ATTRIBUTE_TYPE

procedure apply_attribute_type (
    sds             : in    Pcte.object_reference;
    attribute_type  : in    Pcte.type_name_in_sds;
    pcte_type       : in    Pcte.type_name_in_sds;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.3 SDS_APPLY_LINK_TYPE

procedure apply_link_type (
    sds             : in    Pcte.object_reference;
    link_type       : in    Pcte.type_name_in_sds;
    object_type     : in    Pcte.type_name_in_sds;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 10.2.4 SDS_CREATE_BOOLEAN_ATTRIBUTE_TYPE

procedure create_boolean_attribute_type (
    sds                 : in        Pcte.object_reference;
    duplication         : in        Pcte_sds.duplication;
    new_type            : in out    Pcte.type_name_in_sds;
    new_type_length     : out       Pcte.natural;
    local_name          : in        Pcte.name := "";
    initial_value       : in        Pcte.boolean := FALSE;
    status              : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.5 SDS_CREATE_DESIGNATION_LINK_TYPE

procedure create_designation_link_type (
    sds                 : in        Pcte.object_reference;
    lower_bound         : in        Pcte.natural;
    upper_bound         : in        Pcte.natural;
    duplication         : in        Pcte_sds.duplication;
    key_types           : in        Pcte.type_names_in_sds.sequence;
    new_type            : in out    Pcte.type_name_in_sds;
    new_type_length     : out       Pcte.natural;
    local_name          : in        Pcte.name := "";
    status              : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.6 SDS_CREATE_ENUMERAL_TYPE

procedure create_enumeral_type (
    sds                 : in        Pcte.object_reference;
    new_type            : in out    Pcte.type_name_in_sds;
    new_type_length     : out       Pcte.natural;
    local_name          : in        Pcte.name := "";
    status              : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.7 SDS_CREATE_ENUMERATION_ATTRIBUTE_TYPE

procedure create_enumeration_attribute_type (
    sds                 : in        Pcte.object_reference;
    duplication         : in        Pcte_sds.duplication;
    values              : in        Pcte.type_names_in_sds.sequence;
    new_type            : in out    Pcte.type_name_in_sds;
    new_type_length     : out       Pcte.natural;
    local_name          : in        Pcte.name := "";
    initial_value       : in        Pcte.natural := 0;
    status              : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 10.2.8 SDS_CREATE_FLOAT_ATTRIBUTE_TYPE

procedure create_float_attribute_type (
    sds                 : in      Pcte.object_reference;
    duplication         : in      Pcte_sds.duplication;
    new_type            : in out  Pcte.type_name_in_sds;
    new_type_length     : out     Pcte.natural;
    local_name          : in      Pcte.name := "";
    initial_value       : in      Pcte.float := 0.0;
    status              : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.9 SDS_CREATE_INTEGER_ATTRIBUTE_TYPE

procedure create_integer_attribute_type (
    sds                 : in      Pcte.object_reference;
    duplication         : in      Pcte_sds.duplication;
    new_type            : in out  Pcte.type_name_in_sds;
    new_type_length     : out     Pcte.natural;
    local_name          : in      Pcte.name := "";
    initial_value       : in      Pcte.integer := 0;
    status              : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.10 SDS_CREATE_NATURAL_ATTRIBUTE_TYPE

procedure create_natural_attribute_type (
    sds                 : in      Pcte.object_reference;
    duplication         : in      Pcte_sds.duplication;
    new_type            : in out  Pcte.type_name_in_sds;
    new_type_length     : out     Pcte.natural;
    local_name          : in      Pcte.name := "";
    initial_value       : in      Pcte.natural := 0;
    status              : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.11 SDS_CREATE_OBJECT_TYPE

procedure create_object_type (
    sds                 : in      Pcte.object_reference;
    parents             : in      Pcte.type_names_in_sds.sequence;
    new_type            : in out  Pcte.type_name_in_sds;
    new_type_length     : out     Pcte.natural;
    local_name          : in      Pcte.name := "";
    status              : in      Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 10.2.12 SDS_CREATE_RELATIONSHIP_TYPE

procedure create_relationship_type (
    sds                       : in      Pcte.object_reference;
    forward_properties        : in      Pcte_sds.link_type_properties;
    reverse_properties        : in      Pcte_sds.link_type_properties;
    new_forward_type          : in out  Pcte.type_name_in_sds;
    new_reverse_type          : in out  Pcte.type_name_in_sds;
    new_forward_type_length   : out     Pcte.natural;
    new_reverse_type_length   : out     Pcte.natural;
    forward_local_name        : in      Pcte.name := "";
    reverse_local_name        : in      Pcte.name := "";
    status                    : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.13 SDS_CREATE_STRING_ATTRIBUTE_TYPE

procedure create_string_attribute_type (
    sds               : in      Pcte.object_reference;
    duplication       : in      Pcte_sds.duplication;
    new_type          : in out  Pcte.type_name_in_sds;
    new_type_length   : out     Pcte.natural;
    local_name        : in      Pcte.name := "";
    initial_value     : in      Pcte.string := "";
    status            : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.14 SDS_CREATE_TIME_ATTRIBUTE_TYPE

procedure create_time_attribute_type (
    sds               : in      Pcte.object_reference;
    duplication       : in      Pcte_sds.duplication;
    new_type          : in out  Pcte.type_name_in_sds;
    new_type_length   : out     Pcte.natural;
    local_name        : in      Pcte.name := "";
    initial_value     : in      Pcte.calendar.time := DEFAULT_TIME;
    status            : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.15 SDS_GET_NAME

function get_name (
    sds         : Pcte.object_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.name;

-- 10.2.16 SDS_IMPORT_ATTRIBUTE_TYPE

procedure import_attribute_type (
    to_sds          : in    Pcte.object_reference;
    from_sds        : in    Pcte.object_reference;
    pcte_type       : in    Pcte.type_name_in_sds;
    local_name      : in    Pcte.name := "";
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 10.2.17 SDS_IMPORT_ENUMERAL_TYPE

procedure import_enumeral_type (
    to_sds          : in    Pcte.object_reference;
    from_sds        : in    Pcte.object_reference;
    pcte_type       : in    Pcte.type_name_in_sds;
    local_name      : in    Pcte.name := "";
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.18 SDS_IMPORT_LINK_TYPE

procedure import_link_type (
    to_sds          : in    Pcte.object_reference;
    from_sds        : in    Pcte.object_reference;
    pcte_type       : in    Pcte.type_name_in_sds;
    local_name      : in    Pcte.name := "";
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.19 SDS_IMPORT_OBJECT_TYPE

procedure import_object_type (
    to_sds          : in    Pcte.object_reference;
    from_sds        : in    Pcte.object_reference;
    pcte_type       : in    Pcte.type_name_in_sds;
    local_name      : in    Pcte.name := "";
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.20 SDS_INITIALIZE

procedure initialize (
    sds     : in    Pcte.object_reference;
    name    : in    Pcte.name;
    status  : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.21 SDS_REMOVE

procedure remove (
    sds     : in    Pcte.object_reference;
    status  : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.22 SDS_REMOVE_DESTINATION

procedure remove_destination (
    sds             : in    Pcte.object_reference;
    link_type       : in    Pcte.type_name_in_sds;
    object_type     : in    Pcte.type_name_in_sds;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 10.2.23 SDS_REMOVE_TYPE

procedure remove_type (
    sds        : in    Pcte.object_reference;
    pcte_type  : in    Pcte.type_name_in_sds;
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.24 SDS_SET_ENUMERAL_TYPE_IMAGE

procedure set_enumeral_type_image (
    sds        : in    Pcte.object_reference;
    pcte_type  : in    Pcte.type_name_in_sds;
    image      : in    Pcte.text := "";
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.25 SDS_SET_TYPE_MODES

--  The effect of providing the optional parameter usage_mode but not the optional parameter
--  export_mode is obtained by the subprogram Pcte_sds.set_usage_modes. The effect of
--  providing export_mode but not usage_mode is obtained by the subprogram
--  Pcte_sds.set_export_modes. The effect of providing both usage_mode and export_mode
--  is obtained by the subprogram Pcte_sds.set_type_modes. The effect of providing neither
--  usage_mode nor export_mode is null and so no corresponding subprogram is required.

procedure set_usage_modes (
    sds            : in    Pcte.object_reference;
    pcte_type      : in    Pcte.type_name_in_sds;
    usage_mode     : in    Pcte_sds.definition_mode_values;
    status         : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set_export_modes (
    sds            : in    Pcte.object_reference;
    pcte_type      : in    Pcte.type_name_in_sds;
    export_mode    : in    Pcte_sds.definition_mode_values;
    status         : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure set_type_modes (
    sds            : in    Pcte.object_reference;
    pcte_type      : in    Pcte.type_name_in_sds;
    usage_mode     : in    Pcte_sds.definition_mode_values;
    export_mode    : in    Pcte_sds.definition_mode_values;
    status         : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.26 SDS_SET_TYPE_NAME

procedure set_type_name (
    sds            : in    Pcte.object_reference;
    pcte_type      : in    Pcte.type_name_in_sds;
    local_name     : in    Pcte.name := "";
    status         : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 10.2.27 SDS_UNAPPLY_ATTRIBUTE_TYPE

procedure unapply_attribute_type (
    sds             : in    Pcte.object_reference;
    attribute_type  : in    Pcte.type_name_in_sds;
    pcte_type       : in    Pcte.type_name_in_sds;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.2.28 SDS_UNAPPLY_LINK_TYPE

procedure unapply_link_type (
    sds             : in    Pcte.object_reference;
    link_type       : in    Pcte.type_name_in_sds;
    object_type     : in    Pcte.type_name_in_sds;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);
```

## 10.3 Usage operations

```
-- 10.3.1 SDS_GET_ATTRIBUTE_TYPE_PROPERTIES

-- If the abstract operation returns an enumeration value type in value_type then
-- properties.type_is is set to ENUMERAL_TYPE and properties.enumeral_types
-- contains the sequence of enumeral type nominators.

procedure get_attribute_type_properties (
    sds         : in    Pcte.object_reference;
    pcte_type   : in    Pcte.type_name_in_sds;
    properties  : out   Pcte_sds.attribute_type_properties;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 10.3.2 SDS_GET_ENUMERAL_TYPE_IMAGE

function get_enumeral_type_image (
    sds         : Pcte.object_reference;
    pcte_type   : Pcte.type_name_in_sds;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.text;

-- 10.3.3 SDS_GET_ENUMERAL_TYPE_POSITION

function get_enumeral_type_position (
    sds         : Pcte.object_reference;
    type1       : Pcte.type_name_in_sds;
    type2       : Pcte.type_name_in_sds;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.natural;
```

```
-- 10.3.4 SDS_GET_LINK_TYPE_PROPERTIES

procedure get_link_type_properties (
    sds                 : in      Pcte.object_reference;
    pcte_type           : in      Pcte.type_name_in_sds;
    properties          : in out  Pcte_sds.link_type_properties;
    pcte_reverse        : in out  Pcte.type_name_in_sds;
    reverse_type_length : out     Pcte.natural;
    status              : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.3.5 SDS_GET_OBJECT_TYPE_PROPERTIES

procedure get_object_type_properties (
    sds        : in      Pcte.object_reference;
    pcte_type  : in      Pcte.type_name_in_sds;
    properties : in out  Pcte_sds.object_type_properties;
    status     : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.3.6 SDS_GET_TYPE_KIND

function get_type_kind (
    sds        : Pcte.object_reference;
    pcte_type  : Pcte.type_name_in_sds;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte_sds.type_kind;

-- 10.3.7 SDS_GET_TYPE_MODES

procedure get_type_modes (
    sds                 : in      Pcte.object_reference;
    pcte_type           : in      Pcte.type_name_in_sds;
    usage_mode          : out     Pcte_sds.definition_mode_values;
    export_mode         : out     Pcte_sds.definition_mode_values;
    max_usage_mode      : out     Pcte_sds.definition_mode_values;
    status              : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.3.8 SDS_GET_TYPE_NAME

function get_type_name (
    sds        : Pcte.object_reference;
    pcte_type  : Pcte.type_name_in_sds;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.name;

-- 10.3.9 SDS_SCAN_ATTRIBUTE_TYPE

procedure scan_attribute_type (
    sds                 : in      Pcte.object_reference;
    pcte_type           : in      Pcte.type_name_in_sds;
    scanning_kind       : in      Pcte_sds.attribute_scan_kind;
    types               : in out  Pcte.type_references.sequence;
    status              : in      Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 10.3.10 SDS_SCAN_ENUMERAL_TYPE

procedure scan_enumeral_type (
    sds        : in        Pcte.object_reference;
    pcte_type  : in        Pcte.type_name_in_sds;
    types      : in out    Pcte.type_references.sequence;
    status     : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 10.3.11 SDS_SCAN_LINK_TYPE

procedure scan_link_type (
    sds              : in        Pcte.object_reference;
    pcte_type        : in        Pcte.type_name_in_sds;
    scanning_kind    : in        Pcte_sds.link_scan_kind;
    types            : in out    Pcte.type_references.sequence;
    status           : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 10.3.12 SDS_SCAN_OBJECT_TYPE

procedure scan_object_type (
    sds              : in        Pcte.object_reference;
    pcte_type        : in        Pcte.type_name_in_sds;
    scanning_kind    : in        Pcte_sds.object_scan_kind;
    types            : in out    Pcte.type_references.sequence;
    status           : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 10.3.13 SDS_SCAN_TYPES

procedure scan_types (
    sds     : in        Pcte.object_reference;
    kind    : in        Pcte_sds.type_kind;
    types   : in out    Pcte.type_references.sequence;
    status  : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure scan_types (
    sds     : in        Pcte.object_reference;
    types   : in out    Pcte.type_references.sequence;
    status  : in        Pcte_error.handle := EXCEPTION_ONLY);
```

## 10.4 Working schema operations

```
-- 10.4.1 WS_GET_ATTRIBUTE_TYPE_PROPERTIES

procedure get_attribute_type_properties (
    pcte_type  : in    Pcte.type_reference;
    properties : out   Pcte_sds.attribute_type_properties;
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 10.4.2 WS_GET_ENUMERAL_TYPE_IMAGE

function get_enumeral_type_image (
    pcte_type : Pcte.type_reference;
    status    : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.text;

-- 10.4.3 WS_GET_ENUMERAL_TYPE_POSITION

function get_enumeral_type_position (
    type1     : Pcte.type_reference;
    type2     : Pcte.type_reference;
    status    : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.natural;

-- 10.4.4 WS_GET_LINK_TYPE_PROPERTIES

procedure get_link_type_properties (
    pcte_type     : in      Pcte.type_reference;
    properties    : in out  Pcte_sds.link_type_properties;
    pcte_reverse  : in out  Pcte.type_reference;
    status        : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.4.6 WS_GET_OBJECT_TYPE_PROPERTIES

procedure get_object_type_properties (
    pcte_type : in      Pcte.type_reference;
    properties : in out Pcte_sds.object_type_properties;
    status    : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.4.7 WS_GET_TYPE_KIND

function get_type_kind (
    pcte_type : Pcte.type_reference;
    status    : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte_sds.type_kind;

-- 10.4.8 WS_GET_TYPE_MODES

function get_type_modes (
    pcte_type : Pcte.type_reference;
    status    : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte_sds.definition_mode_values;

-- 10.4.9 WS_GET_TYPE_NAME

function get_type_name (
    pcte_type : Pcte.type_reference;
    status    : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.name;
```

```
-- 10.4.10 WS_SCAN_ATTRIBUTE_TYPE

procedure scan_attribute_type (
    pcte_type       : in      Pcte.type_reference;
    scanning_kind   : in      Pcte_sds.attribute_scan_kind;
    types           : in out  Pcte.type_references.sequence;
    status          : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.4.11 WS_SCAN_ENUMERAL_TYPE

procedure scan_enumeral_type (
    pcte_type : in      Pcte.type_reference;
    types     : in out  Pcte.type_references.sequence;
    status    : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.4.12 WS_SCAN_LINK_TYPE

procedure scan_link_type (
    pcte_type       : in      Pcte.type_reference;
    scanning_kind   : in      Pcte_sds.link_scan_kind;
    types           : in out  Pcte.type_references.sequence;
    status          : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.4.13 WS_SCAN_OBJECT_TYPE

procedure scan_object_type (
    pcte_type       : in      Pcte.type_reference;
    scanning_kind   : in      Pcte_sds.object_scan_kind;
    types           : in out  Pcte.type_references.sequence;
    status          : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 10.4.14 WS_SCAN_TYPES

procedure scan_types (
    kind    : in      Pcte_sds.type_kind;
    types   : in out  Pcte.type_references.sequence;
    status  : in      Pcte_error.handle := EXCEPTION_ONLY);

procedure scan_types (
    types   : in out  Pcte.type_references.sequence;
    status  : in      Pcte_error.handle := EXCEPTION_ONLY);

end Pcte_sds;
```



# 11 Volumes, devices, and archives

## 11.1 Volume, device, and archive datatypes

```
-- See the beginning of the packages Pcte_archive, Pcte_device, and Pcte_volume.
```

## 11.2 Volume, device, and archive operations

```
with Pcte, Pcte_error, Pcte_discretionary;
package Pcte_archive is
    use Pcte, Pcte_error;
    type archive_identifier is new Pcte.natural;
    -- Pcte_archive.archive_identifier corresponds to the PCTE datatype Archive_identifier.
    type archive_status is (PARTIAL, COMPLETE);
    -- Pcte_archive.archive_status corresponds to the PCTE datatype Archive_status.

    -- 11.2.1 ARCHIVE_CREATE
    procedure create (
        archive_identifier      : in      Pcte.natural;
        on_same_volume_as       : in      Pcte.object_reference;
        access_mask             : in      Pcte_discretionary.object.atomic_access_rights;
        new_archive             : in out  Pcte.object_reference;
        status                  : in      Pcte_error.handle := EXCEPTION_ONLY);

    -- 11.2.2 ARCHIVE_REMOVE
    procedure remove (
        archive    : in    Pcte.object_reference;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 11.2.3 ARCHIVE_RESTORE
    -- ARCHIVE_RESTORE is mapped to two overloaded procedures Pcte_archive.restore
    -- according to whether the value of the parameter scope is Object_designators (first
    -- procedure) or ALL (second procedure).
    procedure restore (
        device                  : in      Pcte.object_reference;
        archive                 : in      Pcte.object_reference;
        objects                 : in      Pcte.object_references.sequence;
        on_same_volume_as       : in      Pcte.object_reference;
        restoring_status        : out     Pcte_archive.archive_status;
        status                  : in      Pcte_error.handle := EXCEPTION_ONLY);
    procedure restore (
        device                  : in      Pcte.object_reference;
        archive                 : in      Pcte.object_reference;
        on_same_volume_as       : in      Pcte.object_reference;
        restoring_status        : out     Pcte_archive.archive_status;
        status                  : in      Pcte_error.handle := EXCEPTION_ONLY);
```

```
    -- 11.2.4 ARCHIVE_SAVE
    procedure save (
        device              : in    Pcte.object_reference;
        archive             : in    Pcte.object_reference;
        objects             : in    Pcte.object_references.sequence;
        archiving_status    : out   Pcte_archive.archive_status;
        status              : in    Pcte_error.handle := EXCEPTION_ONLY);
end Pcte_archive;

with Pcte, Pcte_error, Pcte_contents, Pcte_discretionary;
package Pcte_device is
    use Pcte, Pcte_error;
    type device_identifier is new Pcte.natural;
    --  Pcte_device.device_identifier corresponds to the PCTE datatype Device_identifier.
    -- 11.2.5 DEVICE_CREATE
    procedure create (
        station             : in        Pcte.object_reference;
        device_type         : in        Pcte.type_reference;
        access_mask         : in        Pcte_discretionary.object.atomic_access_rights;
        device_identifier   : in        Pcte.natural;
        device_characteristics : in     Pcte.string;
        new_device          : in out    Pcte.object_reference;
        status              : in        Pcte_error.handle := EXCEPTION_ONLY);
    -- 11.2.6 DEVICE_REMOVE
    procedure remove (
        device      : in    Pcte.object_reference;
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);
    -- 11.2.7 LINK_GET_DESTINATION_ARCHIVE
    --  See 9.2.
    -- 12.2.13 DEVICE_GET_CONTROL
    generic
        type element_type is private;
    function get_control (
        contents    : Pcte_contents.contents_handle;
        operation   : Pcte.natural;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      element_type;
```

```
    -- 12.2.14 DEVICE_SET_CONTROL

    generic
        type element_type is private;
    procedure set_control (
        contents      : in    Pcte_contents.contents_handle;
        operation     : in    Pcte.natural;
        control_data  : in    element_type;
        status        : in    Pcte_error.handle := EXCEPTION_ONLY);
end Pcte_device;


with Pcte, Pcte_error, Pcte_discretionary;
package Pcte_volume is
    use Pcte, Pcte_error;
    type volume_accessibility is (ACCESSIBLE, INACCESSIBLE, UNKNOWN);
    -- Pcte_volume.volume_accessibility corresponds to the PCTE datatype
    -- Volume_accessibility.
    type volume_identifier is new Pcte.natural;
    -- Pcte_volume.volume_identifier corresponds to the PCTE datatype Volume_identifier.
    type volume_info is record
        identity    : Pcte_volume.volume_identifier;
        mounted     : Pcte_volume.volume_accessibility;
    end record;
    -- Pcte_volume.volume_info corresponds to the PCTE datatype Volume_info.
    type volume_status is record
        total_blocks        : Pcte.natural;
        free_blocks         : Pcte.natural;
        block_size          : Pcte.natural;
        num_objects         : Pcte.natural;
        volume_identifier   : Pcte_volume.volume_identifier;
    end record;
    -- Pcte_volume.volume_status corresponds to the PCTE datatype Volume_status.


    -- The semantics of the operations of this package are defined in 8.2.8.
    package volume_infos is
        type sequence is limited private;
        -- Pcte_volume.volume_infos.sequence corresponds to the PCTE datatype
        -- Volume_infos.
```

```
function get (
    list        : Pcte_volume.volume_infos.sequence;
    index       : Pcte.natural := Pcte.natural'FIRST;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte_volume.volume_info;

procedure insert (
    list        : in out    Pcte_volume.volume_infos.sequence;
    item        : in        Pcte_volume.volume_info;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure replace (
    list        : in out    Pcte_volume.volume_infos.sequence;
    item        : in        Pcte_volume.volume_info;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure append (
    list        : in out    Pcte_volume.volume_infos.sequence;
    item        : in        Pcte_volume.volume_info;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure delete (
    list        : in out    Pcte_volume.volume_infos.sequence;
    index       : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure copy (
    into_list       : in out    Pcte_volume.volume_infos.sequence;
    from_list       : in        Pcte_volume.volume_infos.sequence;
    into_index      : in        Pcte.natural := Pcte.natural'LAST;
    from_index      : in        Pcte.natural := Pcte.natural'FIRST;
    count           : in        Pcte.positive := Pcte.positive'LAST;
    status          : in        Pcte_error.handle := EXCEPTION_ONLY);

function length_of (
    list        : Pcte_volume.volume_infos.sequence;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.natural;

function index_of (
    list        : Pcte_volume.volume_infos.sequence;
    item        : Pcte_volume.volume_info;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.integer;
```

```
    function are_equal (
        first       : Pcte_volume.volume_infos.sequence;
        second      : Pcte_volume.volume_infos.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.boolean;

    procedure normalize (
        list    : in out    Pcte_volume.volume_infos.sequence;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list    : in out    Pcte_volume.volume_infos.sequence;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end volume_infos;

-- 11.2.8 VOLUME_CREATE

procedure create (
    device                  : in        Pcte.object_reference;
    volume_identifier       : in        Pcte.natural;
    access_mask             : in        Pcte_discretionary.object.atomic_access_rights;
    volume_characteristics  : in        Pcte.string;
    new_volume              : in out    Pcte.object_reference;
    status                  : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 11.2.9 VOLUME_DELETE

procedure delete (
    volume    : in     Pcte.object_reference;
    status    : in     Pcte_error.handle := EXCEPTION_ONLY);

-- 11.2.10 VOLUME_GET_STATUS

function get_status (
    volume    : Pcte.object_reference;
    status    : Pcte_error.handle := EXCEPTION_ONLY)
    return     Pcte_volume.volume_status;

-- 11.2.11 VOLUME_MOUNT

procedure mount (
    device              : in    Pcte.object_reference;
    volume_identifier   : in    Pcte_volume.volume_identifier;
    read_only           : in    Pcte.boolean;
    status              : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
    -- 11.2.12 VOLUME_UNMOUNT

    procedure unmount (
        volume      : in     Pcte.object_reference;
        status      : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 9.3.20 VOLUME_LIST_OBJECTS

    procedure list_objects (
        volume      : in        Pcte.object_reference;
        types       : in        Pcte.type_references.sequence;
        objects     : in out    Pcte.object_references.sequence;
        status      : in        Pcte_error.handle:= EXCEPTION_ONLY);
end Pcte_volume;


with Pcte, Pcte_error, Pcte_discretionary;

package Pcte_cluster is

    use Pcte, Pcte_error;

    -- 11.3.1 CLUSTER_CREATE

    procedure create (
        on_same_volume_as     : in        Pcte.object_reference;
        cluster_identifier    : in        Pcte.natural;
        access_mask           : in        Pcte_discretionary.object.atomic_access_rights;
        cluster_characteristics : in      Pcte.string;
        new_cluster           : in out    Pcte.object_reference;
        status                : in        Pcte_error.handle := EXCEPTION_ONLY);

    -- 11.3.2 CLUSTER_DELETE

    procedure delete (
        cluster     : in     Pcte.object_reference;
        status      : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 11.3.3 CLUSTER_LIST_OBJECTS

    procedure list_objects (
        cluster     : in        Pcte.object_reference;
        types       : in        Pcte.type_references.sequence
        objects     : in out    Pcte.object_references.sequence
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
end Pcte_cluster;
```

## 12 Files, pipes, and devices

```
with Pcte, Pcte_error;
```

```
package Pcte_contents is

    use Pcte, Pcte_error;
```

## 12.1 File, pipe, and device datatypes

```
    type contents_access_mode is (READ_WRITE, READ_ONLY, WRITE_ONLY,
        APPEND_ONLY);

    -- Pcte_contents.contents_access_mode corresponds to the PCTE datatype
    -- Contents_access_mode.

    type seek_position is (FROM_BEGINNING, FROM_CURRENT, FROM_END);

    -- Pcte_contents.seek_position corresponds to the PCTE datatype Seek_position.

    type pcte_set_position is (AT_BEGINNING, AT_POSITION, AT_END);

    -- Pcte_contents.pcte_set_position corresponds to the PCTE datatype Set_position.

    type positioning_style is (SEQUENTIAL, DIRECT, SEEK);

    -- Pcte_contents.positioning_style corresponds to the PCTE datatype Positioning_style.

    type position_handle is limited private;

    -- Pcte_contents.position_handle corresponds to the PCTE datatype Position_handle.

    type contents_handle is limited private;

    -- Pcte_contents.contents_handle corresponds to the PCTE datatype Contents_handle.
```

## 12.2 File, pipe, and device operations

```
    -- The operations which return values of type Contents_handle can give rise to the binding-
    -- defined error condition CONTENTS_HANDLE_IS_OPEN(contents).

    -- 12.2.1 CONTENTS_CLOSE

    procedure close (
        contents : in     Pcte_contents.contents_handle;
        status   : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 12.2.2 CONTENTS_GET_HANDLE_FROM_KEY

    procedure get_handle_from_key (
        open_object_key : in     Pcte.natural;
        contents        : in out Pcte_contents.contents_handle;
        status          : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 12.2.3 CONTENTS_GET_KEY_FROM_HANDLE

    function get_key_from_handle (
        contents : Pcte_contents.contents_handle;
        status   : Pcte_error.handle := EXCEPTION_ONLY)
        return     Pcte.natural;
```

```
-- 12.2.4 CONTENTS_GET_POSITION

procedure get_position (
    contents   : in     Pcte_contents.contents_handle;
    position   : out    Pcte_contents.position_handle;
    status     : in     Pcte_error.handle := EXCEPTION_ONLY);

-- 12.2.5 CONTENTS_HANDLE_DUPLICATE

procedure handle_duplicate (
    contents       : in      Pcte_contents.contents_handle;
    new_key        : in      Pcte.natural;
    inheritable    : in      Pcte.boolean;
    new_contents   : in out  Pcte_contents.contents_handle;
    status         : in      Pcte_error.handle := EXCEPTION_ONLY);

procedure handle_duplicate (
    contents       : in      Pcte_contents.contents_handle;
    inheritable    : in      Pcte.boolean;
    new_contents   : in out  Pcte_contents.contents_handle;
    status         : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 12.2.6 CONTENTS_OPEN

procedure open (
    object             : in      Pcte.object_reference;
    opening_mode       : in      Pcte_contents.contents_access_mode;
    non_blocking_io    : in      Pcte.boolean;
    inheritable        : in      Pcte.boolean;
    contents           : in out  Pcte_contents.contents_handle;
    status             : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 12.2.7 CONTENTS_READ

function read (
    contents   : Pcte_contents.contents_handle;
    size       : Pcte.natural;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.string;

-- 12.2.8 CONTENTS_SEEK

procedure seek (
    contents       : in     Pcte_contents.contents_handle;
    offset         : in     Pcte.integer;
    whence         : in     Pcte_contents.seek_position;
    new_position   : out    Pcte.natural;
    status         : in     Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 12.2.9 CONTENTS_SET_POSITION

procedure set_position (
    contents          : in    Pcte_contents.contents_handle;
    position_handle   : in    Pcte_contents.position_handle;
    set_mode          : in    Pcte_contents.pcte_set_position;
    status            : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 12.2.10 CONTENTS_SET_PROPERTIES

procedure set_properties (
    contents      : in    Pcte_contents.contents_handle;
    positioning   : in    Pcte_contents.positioning_style;
    status        : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 12.2.11 CONTENTS_TRUNCATE

procedure truncate (
    contents  : in    Pcte_contents.contents_handle;
    status    : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 12.2.12 CONTENTS_WRITE

procedure write (
    contents      : in    Pcte_contents.contents_handle;
    data          : in    Pcte.string;
    actual_size   : out   Pcte.natural;
    status        : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 12.2.13 DEVICE_GET_CONTROL

--   See 11.2.

-- 12.2.14 DEVICE_SET_CONTROL

--   See 11.2.

-- 18.3.1 CONTENTS_COPY_FROM_FOREIGN_SYSTEM

procedure copy_from_foreign_system (
    file                : in    Pcte.object_reference;
    foreign_system      : in    Pcte.object_reference;
    foreign_name        : in    Pcte.string;
    foreign_parameters  : in    Pcte.string;
    status              : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 18.3.2 CONTENTS_COPY_TO_FOREIGN_SYSTEM

procedure copy_to_foreign_system (
    file                : in    Pcte.object_reference;
    foreign_system      : in    Pcte.object_reference;
    foreign_name        : in    Pcte.string;
    foreign_parameters  : in    Pcte.string;
    status              : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
private
    implementation-defined
end Pcte_contents;
```

# 13 Process execution

```
with Pcte, Pcte_error, Pcte_mandatory, Pcte_discretionary, SYSTEM;
package Pcte_process is
    use Pcte, Pcte_error, Pcte.reference;
```

## 13.1 Process execution datatypes

```
    type initial_status is (RUNNING, STOPPED, SUSPENDED);
    -- Pcte_process.initial_status corresponds to the PCTE datatype Initial_status.
    EXIT_SUCCESS :          constant Pcte.integer := implementation-defined;
    EXIT_ERROR :            constant Pcte.integer := implementation-defined;
    FORCED_TERMINATION :    constant Pcte.integer := implementation-defined;
    SYSTEM_FAILURE :        constant Pcte.integer := implementation-defined;
    ACTIVITY_ABORTED :      constant Pcte.integer := implementation-defined;
    -- These constants define the possible termination status of a process.
    subtype address is SYSTEM.ADDRESS;
    -- Pcte_process.address corresponds to the PCTE datatype Address.
    type profile_handle is limited private;
    -- Pcte_process.profile_handle corresponds to the PCTE datatype Profile_handle.

    -- The semantics of the operations of this package are defined in 8.2.8.
    package profile_buffer is
        type sequence is limited private;
        -- Pcte_process.profile_buffer.sequence corresponds to the PCTE datatype Buffer.
        function get (
            list        : Pcte_process.profile_buffer.sequence;
            index       : Pcte.natural := Pcte.natural'FIRST;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return        Pcte.natural;
```

```
procedure insert (
    list        : in out    Pcte_process.profile_buffer.sequence;
    item        : in        Pcte.natural;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure replace (
    list        : in out    Pcte_process.profile_buffer.sequence;
    item        : in        Pcte.natural;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure append (
    list        : in out    Pcte_process.profile_buffer.sequence;
    item        : in        Pcte.natural;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure delete (
    list        : in out    Pcte_process.profile_buffer.sequence;
    index       : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure copy (
    into_list       : in out    Pcte_process.profile_buffer.sequence;
    from_list       : in        Pcte_process.profile_buffer.sequence;
    into_index      : in        Pcte.natural := Pcte.natural'LAST;
    from_index      : in        Pcte.natural := Pcte.natural'FIRST;
    count           : in        Pcte.positive := Pcte.positive'LAST;
    status          : in        Pcte_error.handle := EXCEPTION_ONLY);

function length_of (
    list        : Pcte_process.profile_buffer.sequence;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.natural;

function index_of (
    list        : Pcte_process.profile_buffer.sequence;
    item        : Pcte.natural;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.integer;

function are_equal (
    first       : Pcte_process.profile_buffer.sequence;
    second      : Pcte_process.profile_buffer.sequence;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.boolean;

procedure normalize (
    list        : in out    Pcte_process.profile_buffer.sequence;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure discard (
        list        : in out    Pcte_process.profile_buffer.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
private
    implementation-defined
end profile_buffer;

-- The semantics of the operations of this package are defined in 8.2.8.
package names is
    type sequence is limited private;
    -- Pcte_process.names.sequence corresponds to the PCTE datatype Name_sequence.
    function get (
        list        : Pcte_process.names.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.name;
    procedure insert (
        list        : in out    Pcte_process.names.sequence;
        item        : in        Pcte.name;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
    procedure replace (
        list        : in out    Pcte_process.names.sequence;
        item        : in        Pcte.name;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
    procedure append (
        list        : in out    Pcte_process.names.sequence;
        item        : in        Pcte.name;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
    procedure delete (
        list        : in out    Pcte_process.names.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure copy (
        into_list      : in out   Pcte_process.names.sequence;
        from_list      : in       Pcte_process.names.sequence;
        into_index     : in       Pcte.natural := Pcte.natural'LAST;
        from_index     : in       Pcte.natural := Pcte.natural'FIRST;
        count          : in       Pcte.positive := Pcte.positive'LAST;
        status         : in       Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list       : Pcte_process.names.sequence;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.natural;

    function index_of (
        list       : Pcte_process.names.sequence;
        item       : Pcte.name;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.integer;

    function are_equal (
        first      : Pcte_process.names.sequence;
        second     : Pcte_process.names.sequence;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte.boolean;

    procedure normalize (
        list       : in out   Pcte_process.names.sequence;
        status     : in       Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list       : in out   Pcte_process.names.sequence;
        status     : in       Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end names;
```

## 13.2 Process execution

```
-- 13.2.1 PROCESS_CREATE

procedure create  (
    static_context     : in       Pcte.object_reference;
    process_type       : in       Pcte.type_reference;
    implicit_deletion  : in       Pcte.boolean;
    access_mask        : in       Pcte_discretionary.object.atomic_access_rights;
    new_process        : in out   Pcte.object_reference;
    parent             : in       Pcte.object_reference := CURRENT_PROCESS;
    site               : in       Pcte.object_reference := LOCAL_EXECUTION_SITE;
    status             : in       Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 13.2.2 PROCESS_CREATE_AND_START

procedure create_and_start (
    static_context      : in        Pcte.object_reference;
    arguments           : in        Pcte.string;
    environment         : in        Pcte.string;
    implicit_deletion   : in        Pcte.boolean;
    access_mask         : in        Pcte_discretionary.object.atomic_access_rights;
    new_process         : in out    Pcte.object_reference;
    site                : in        Pcte.object_reference := LOCAL_EXECUTION_SITE;
    status              : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.3 PROCESS_GET_WORKING_SCHEMA

procedure get_working_schema (
    sds_sequence : in out   Pcte_process.names.sequence;
    process      : in       Pcte.object_reference := CURRENT_PROCESS;
    status       : in       Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.4 PROCESS_INTERRUPT_OPERATION

procedure interrupt_operation (
    process     : in    Pcte.object_reference;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.5 PROCESS_RESUME

procedure resume (
    process     : in    Pcte.object_reference;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.6 PROCESS_SET_ALARM

procedure set_alarm (
    duration    : in    Pcte.natural;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.7 PROCESS_SET_FILE_SIZE_LIMIT

procedure set_file_size_limit (
    fslimit     : in    Pcte.natural;
    process     : in    Pcte.object_reference := CURRENT_PROCESS;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.8 PROCESS_SET_OPERATION_TIME_OUT

procedure set_operation_time_out (
    duration    : in    Pcte.natural;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 13.2.9 PROCESS_SET_PRIORITY

procedure set_priority (
    priority    : in    Pcte.natural;
    process     : in    Pcte.object_reference := CURRENT_PROCESS;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.10 PROCESS_SET_REFERENCED_OBJECT

procedure set_referenced_object (
    reference_name  : in    Pcte.actual_key;
    object          : in    Pcte.object_reference;
    process         : in    Pcte.object_reference := CURRENT_PROCESS;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.11 PROCESS_SET_TERMINATION_STATUS

procedure set_termination_status (
    termination_status  : in    Pcte.integer;
    status              : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.12 PROCESS_SET_WORKING_SCHEMA

procedure set_working_schema (
    sds_sequence : in   Pcte_process.names.sequence;
    process      : in   Pcte.object_reference := CURRENT_PROCESS;
    status       : in   Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.13 PROCESS_START

procedure start (
    process         : in    Pcte.object_reference;
    arguments       : in    Pcte.string;
    environment     : in    Pcte.string;
    initial_status  : in    Pcte_process.initial_status;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure start (
    process         : in    Pcte.object_reference;
    arguments       : in    Pcte.string;
    environment     : in    Pcte.string;
    site            : in    Pcte.object_reference;
    initial_status  : in    Pcte_process.initial_status;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.14 PROCESS_SUSPEND

procedure suspend (
    process     : in    Pcte.object_reference := CURRENT_PROCESS;
    alarm       : in    Pcte.natural := 0;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 13.2.15 PROCESS_TERMINATE

procedure terminate_process (
    process             : in    Pcte.object_reference := CURRENT_PROCESS;
    termination_status  : in    Pcte.integer := FORCED_TERMINATION;
    status              : in    Pcte_error.handle := EXCEPTION_ONLY);

--  The name of the procedure is modified because 'terminate' is an Ada reserved word.

-- 13.2.16 PROCESS_UNSET_REFERENCED_OBJECT

procedure unset_referenced_object (
    reference_name  : in    Pcte.actual_key;
    process         : in    Pcte.object_reference := CURRENT_PROCESS;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.17 PROCESS_WAIT_FOR_ANY_CHILD

procedure wait_for_any_child (
    termination_status  : out   Pcte.integer;
    child               : out   Pcte.natural;
    status              : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.2.18 PROCESS_WAIT_FOR_CHILD

procedure wait_for_child (
    child               : in    Pcte.object_reference;
    termination_status  : out   Pcte.integer;
    status              : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 22.3.1 PROCESS_SET_CONSUMER_IDENTITY

procedure set_consumer_identity (
    group  : in     Pcte.object_reference;
    status : in     Pcte_error.handle := EXCEPTION_ONLY);

-- 22.3.2 PROCESS_UNSET_CONSUMER_IDENTITY

procedure unset_consumer_identity (
    status : in     Pcte_error.handle := EXCEPTION_ONLY);
```

## 13.3 Security operations

```
-- 13.3.1 PROCESS_ADOPT_USER_GROUP

procedure adopt_user_group (
    user_group  : in    Pcte.object_reference;
    process     : in    Pcte.object_reference := CURRENT_PROCESS;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 13.3.2 PROCESS_GET_DEFAULT_ACL

procedure get_default_acl (
    acl     : in out    Pcte_discretionary.object.acl_entries.sequence;
    status  : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 13.3.3 PROCESS_GET_DEFAULT_OWNER

function get_default_owner (
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte_discretionary.group_identifier;

-- 13.3.4 PROCESS_SET_ADOPTABLE_FOR_CHILD

procedure set_adoptable_for_child (
    user_group    : in    Pcte.object_reference;
    adoptability  : in    Pcte.boolean;
    process       : in    Pcte.object_reference := CURRENT_PROCESS;
    status        : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.3.5 PROCESS_SET_DEFAULT_ACL_ENTRY

procedure set_default_acl_entry (
    group     : in    Pcte_discretionary.group_identifier;
    modes     : in    Pcte_discretionary.object.atomic_access_rights;
    process   : in    Pcte.object_reference := CURRENT_PROCESS;
    status    : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.3.6 PROCESS_SET_DEFAULT_OWNER

procedure set_default_owner (
    group     : in    Pcte_discretionary.group_identifier;
    process   : in    Pcte.object_reference := CURRENT_PROCESS;
    status    : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 13.3.7 PROCESS_SET_USER

procedure set_user (
    user        : in    Pcte.object_reference;
    user_group  : in    Pcte.object_reference;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 20.4.1 PROCESS_SET_CONFIDENTIALITY_LABEL

procedure set_confidentiality_label (
    confidentiality_label : in    Pcte_mandatory.security_label;
    process               : in    Pcte.object_reference := CURRENT_PROCESS;
    status                : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 20.4.2 PROCESS_SET_FLOATING_CONFIDENTIALITY_LEVEL

procedure set_floating_confidentiality_level (
    floating_mode    : in    Pcte_mandatory.floating_level;
    process          : in    Pcte.object_reference := CURRENT_PROCESS;
    status           : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 20.4.3 PROCESS_SET_FLOATING_INTEGRITY_LEVEL

procedure set_floating_integrity_level (
    floating_mode    : in    Pcte_mandatory.floating_level;
    process          : in    Pcte.object_reference := CURRENT_PROCESS;
    status           : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 20.4.4 PROCESS_SET_INTEGRITY_LABEL

procedure set_integrity_label (
    integrity_label  : in    Pcte_mandatory.security_label;
    process          : in    Pcte.object_reference := CURRENT_PROCESS;
    status           : in    Pcte_error.handle := EXCEPTION_ONLY);
```

## 13.4 Profiling operations

```
-- 13.4.1 PROCESS_PROFILING_OFF

procedure profiling_off (
    handle : in      Pcte_process.profile_handle;
    buffer : in out  Pcte_process.profile_buffer.sequence;
    status : in      Pcte_error.handle := EXCEPTION_ONLY);

-- 13.4.2 PROCESS_PROFILING_ON

procedure profiling_on (
    start      : in    Pcte_process.address;
    pcte_end   : in    Pcte_process.address;
    count      : in    Pcte.natural;
    handle     : out   Pcte_process.profile_handle;
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);
```

## 13.5 Monitoring operations

```
-- 13.5.1 PROCESS_ADD_BREAKPOINT

procedure add_breakpoint (
    process      : in    Pcte.object_reference;
    breakpoint   : in    Pcte_process.address;
    status       : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
    -- 13.5.2 PROCESS_CONTINUE

    procedure continue (
        process     : in    Pcte.object_reference;
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 13.5.3 PROCESS_PEEK

    generic
        type process_data is private;
    function peek (
        process     : Pcte.object_reference
        address     : Pcte_process.address;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      process_data;

    -- 13.5.4 PROCESS_POKE

    generic
        type process_data is private;
    procedure poke (
        process     : in    Pcte.object_reference;
        address     : in    Pcte_process.address;
        value       : in    process_data;
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 13.5.5 PROCESS_REMOVE_BREAKPOINT

    procedure remove_breakpoint (
        process         : in    Pcte.object_reference;
        breakpoint      : in    Pcte_process.address;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 13.5.6 PROCESS_WAIT_FOR_BREAKPOINT

    procedure wait_for_breakpoint (
        process         : in    Pcte.object_reference;
        breakpoint      : out   Pcte_process.address;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);
private
    implementation-defined
end Pcte_process;
```

## 14 Message queues

```
with Pcte, Pcte_error;
package Pcte_message is
    use Pcte_error;
```

## 14.1 Message queue datatypes

```
type standard_message_type is (INTERRUPT_MSG, QUIT_MSG, FINISH_MSG,
    SUSPEND_MSG, END_MSG, ABORT_MSG, DEADLOCK_MSG, WAKE_MSG);

-- Pcte_message.standard_message_type corresponds to the PCTE datatype
-- Standard_message_type.

type notification_message_type is (MODIFICATION_MSG, CHANGE_MSG,
        DELETE_MSG, MOVE_MSG, NOT_ACCESSIBLE_MSG, LOST_MSG);

-- Pcte_message.notification_message_type corresponds to the PCTE datatype
-- Notification_message_type.

type implementation_defined_message_type is new Pcte.natural;

-- Pcte_message.implementation_defined_message_type corresponds to the PCTE
-- datatype Implementation_defined_message_type.

type undefined_message_type is new Pcte.natural;

-- Pcte_message.undefined_message_type corresponds to the PCTE datatype
-- Undefined_message_type.

type message_type_kind is (STANDARD_MESSAGE,
    NOTIFICATION_MESSAGE, IMPLEMENTATION_DEFINED_MESSAGE,
    UNDEFINED_MESSAGE);

type message_type (
    kind    : Pcte_message.message_type_kind := STANDARD_MESSAGE)
is record
    case kind is
        when STANDARD_MESSAGE =>
            standard    : Pcte_message.standard_message_type;
        when NOTIFICATION_MESSAGE =>
            notification    : Pcte_message.notification_message_type;
        when IMPLEMENTATION_DEFINED_MESSAGE =>
            implementation_defined :
                Pcte_message.implementation_defined_message_type;
        when UNDEFINED_MESSAGE =>
            undefined : Pcte_message.undefined_message_type;
    end case;
end record;

type received_message (
    string_length : Pcte.string_length := 0)
is record
    pcte_type           : Pcte_message.message_type;
    message_received    : Pcte.boolean;
    position            : Pcte.natural;
    data                : Pcte.string(1..string_length);
end record;

-- Pcte_message.received_message corresponds to the PCTE datatype Received_message.
```

```
type receive_mode is (PEEK, NO_WAIT, WAIT);

-- The values of Pcte_message.receive_mode determine the mapping of the operations
-- MESSAGE_PEEK, MESSAGE_RECEIVE_NO_WAIT, and MESSAGE_RECEIVE_
-- WAIT.

subtype send_mode is Pcte_message.receive_mode range NO_WAIT .. WAIT;

-- The values of Pcte_message.send_mode distinguish between calls corresponding to
-- MESSAGE_SEND_NO_WAIT and MESSAGE_SEND_WAIT.


-- The semantics of the operations of this package are defined in 8.2.8.

package message_types is

    type sequence is limited private;

    -- Pcte_message.message_types.sequence corresponds to the PCTE
    -- datatype Message_types.

    function get (
        list        : Pcte_message.message_types.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte_message.message_type;

    procedure insert (
        list        : in out    Pcte_message.message_types.sequence;
        item        : in        Pcte_message.message_type;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure replace (
        list        : in out    Pcte_message.message_types.sequence;
        item        : in        Pcte_message.message_type;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure append (
        list        : in out    Pcte_message.message_types.sequence;
        item        : in        Pcte_message.message_type;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure delete (
        list        : in out    Pcte_message.message_types.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure copy (
        into_list       : in out    Pcte_message.message_types.sequence;
        from_list       : in        Pcte_message.message_types.sequence;
        into_index      : in        Pcte.natural := Pcte.natural'LAST;
        from_index      : in        Pcte.natural := Pcte.natural'FIRST;
        count           : in        Pcte.positive := Pcte.positive'LAST;
        status          : in        Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list        : Pcte_message.message_types.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.natural;

    function index_of (
        list        : Pcte_message.message_types.sequence;
        item        : Pcte_message.message_type;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.integer;

    function are_equal (
        first       : Pcte_message.message_types.sequence;
        second      : Pcte_message.message_types.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte_message.message_types.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte_message.message_types.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end message_types;
```

## 14.2 Message queue operations

```
-- 14.2.1 MESSAGE_DELETE

procedure delete (
    queue       : in    Pcte.object_reference;
    position    : in    Pcte.natural;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 14.2.2 MESSAGE_PEEK

-- The abstract operation MESSAGE_PEEK is mapped to the functions Pcte_message.
-- receive with the parameter mode set to PEEK. The abstract operations
-- MESSAGE_RECEIVE_NO_WAIT and MESSAGE_RECEIVE_WAIT are also mapped
-- to this function, see below. The effect of assigning set of Message_type to the parameter
-- types is achieved by the first overloaded function. The effect of assigning
-- ALL_MESSAGE_TYPES to types is achieved by the second overloaded function.

function receive (
    queue      : Pcte.object_reference;
    types      : Pcte_message.message_types.sequence;
    position   : Pcte.natural := 0;
    mode       : Pcte_message.receive_mode := PEEK;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte_message.received_message;

function receive (
    queue      : Pcte.object_reference;
    position   : Pcte.natural := 0;
    mode       : Pcte_message.receive_mode := PEEK;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte_message.received_message;

-- 14.2.3 MESSAGE_RECEIVE_NO_WAIT

-- This abstract operation is mapped to the functions Pcte_message.receive with the
-- parameter mode set to NO_WAIT.

-- 14.2.4 MESSAGE_RECEIVE_WAIT

-- This abstract operation is mapped to the functions Pcte_message.receive with the
-- parameter mode set to WAIT.

-- 14.2.5 MESSAGE_SEND_NO_WAIT

-- This abstract operation is mapped to the procedure Pcte_message.send with the parameter
-- mode set to NO_WAIT. The parameters message_data and message_type correspond
-- respectively to the DATA and MESSAGE_TYPE fields of the PCTE datatype Message.

procedure send (
    queue          : in   Pcte.object_reference;
    message_data   : in   Pcte.string;
    message_type   : in   Pcte_message.message_type;
    mode           : in   Pcte_message.send_mode := NO_WAIT;
    status         : in   Pcte_error.handle := EXCEPTION_ONLY);

-- 14.2.6 MESSAGE_SEND_WAIT

-- This abstract operation is mapped to the procedure Pcte_message.send with the parameter
-- mode set to WAIT.
```

```
end Pcte_message;

with Pcte, Pcte_error, Pcte_message;
package Pcte_queue is
    use Pcte_error, Pcte_message;

    -- 14.2.7 QUEUE_EMPTY

    procedure empty (
        queue  : in     Pcte.object_reference;
        status : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 14.2.8 QUEUE_HANDLER_DISABLE

    procedure handler_disable (
        queue  : in     Pcte.object_reference;
        status : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 14.2.9 QUEUE_HANDLER_ENABLE

    --  The effect of assigning set of Message_type to the parameter types is achieved by the first
    --  overloaded procedure. The effect of assigning ALL_MESSAGE_TYPES to types is
    --  achieved by the second overloaded procedure.

    generic
        with procedure handler (
            queue  : in     Pcte.object_reference);

    package handlers is

        procedure enable (
            queue  : in     Pcte.object_reference;
            types  : in     Pcte_message.message_types.sequence;
            status : in     Pcte_error.handle := EXCEPTION_ONLY);

        procedure enable (
            queue  : in     Pcte.object_reference;
            status : in     Pcte_error.handle := EXCEPTION_ONLY);

    end handlers;

    -- 14.2.10 QUEUE_RESERVE

    procedure reserve (
        queue  : in     Pcte.object_reference;
        status : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 14.2.11 QUEUE_RESTORE

    procedure restore (
        queue  : in     Pcte.object_reference;
        file   : in     Pcte.object_reference;
        status : in     Pcte_error.handle := EXCEPTION_ONLY);
```

```
    -- 14.2.12 QUEUE_SAVE

    procedure save (
        queue  : in    Pcte.object_reference;
        file   : in    Pcte.object_reference;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 14.2.13 QUEUE_SET_TOTAL_SPACE

    procedure set_total_space (
        queue       : in    Pcte.object_reference;
        total_space : in    Pcte.natural;
        status      : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 14.2.14 QUEUE_UNRESERVE

    procedure unreserve (
        queue  : in    Pcte.object_reference;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);

end Pcte_queue;
```

# 15 Notification

```
with Pcte, Pcte_error, Pcte_message;

package Pcte_notify is

    use Pcte_error;
```

## 15.1 Notification datatypes

```
    type access_event is (MODIFICATION_EVENT, CHANGE_EVENT, DELETE_EVENT,
        MOVE_EVENT);

    --  Pcte_notify.access_event corresponds to the PCTE datatype Access_event.

    type access_events is array (Pcte_notify.access_event) of Pcte.boolean;

    --  Pcte_notify.access_events corresponds to the PCTE datatype Access_events.
```

## 15.2 Notification operations

```
    -- 15.2.1 NOTIFICATION_MESSAGE_GET_KEY

    function get_key (
        message : Pcte_message.received_message;
        status  : Pcte_error.handle := EXCEPTION_ONLY)
        return    Pcte.natural;
```

```
    -- 15.2.2 NOTIFY_CREATE

    procedure create (
        notifier_key   : in    Pcte.natural;
        queue          : in    Pcte.object_reference;
        object         : in    Pcte.object_reference;
        status         : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 15.2.3 NOTIFY_DELETE

    procedure delete (
        notifier_key   : in    Pcte.natural;
        queue          : in    Pcte.object_reference;
        status         : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 15.2.4 NOTIFY_SWITCH_EVENTS

    procedure switch_events (
        notifier_key   : in    Pcte.natural;
        queue          : in    Pcte.object_reference;
        access_events  : in    Pcte_notify.access_events;
        status         : in    Pcte_error.handle := EXCEPTION_ONLY);

end Pcte_notify;
```

# 16 Concurrency and integrity control

```
with Pcte, Pcte_error;

package Pcte_activity is

    use Pcte_error;
```

## 16.1 Concurrency and integrity control datatypes

```
    type activity_class is (UNPROTECTED, PROTECTED, TRANSACTION);

    --  Pcte_activity.activity_class corresponds to the PCTE datatype Activity_class.
```

## 16.2 Concurrency and integrity control operations

```
    -- 16.2.1 ACTIVITY_ABORT

    procedure abort_activity (
        status  : in    Pcte_error.handle := EXCEPTION_ONLY);

    --  The name of the procedure is modified because 'abort' is an Ada reserved word.

    -- 16.2.2 ACTIVITY_END

    procedure end_activity (
        status  : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- The name of the procedure is modified because 'end' is an Ada reserved word.

-- 16.2.3 ACTIVITY_START

procedure start_activity (
    activity_class : in     Pcte_activity.activity_class;
    status         : in     Pcte_error.handle := EXCEPTION_ONLY);

-- The name of the procedure is modified to retain the analogy with Pcte_activity.
-- end_activity.

package lock is

    type lock_set_mode is ( READ_UNPROTECTED, READ_SEMIPROTECTED,
        WRITE_UNPROTECTED, WRITE_SEMIPROTECTED,
        DELETE_UNPROTECTED, DELETE_SEMIPROTECTED, READ_PROTECTED,
        DELETE_PROTECTED, WRITE_PROTECTED, WRITE_TRANSACTIONED,
        DELETE_TRANSACTIONED, READ_DEFAULT, WRITE_DEFAULT,
        DELETE_DEFAULT);

    -- Pcte_activity.lock.lock_set_mode corresponds to the PCTE datatype Lock_set_mode.

    subtype internal_mode is Pcte_activity.lock.lock_set_mode
        range READ_UNPROTECTED .. WRITE_PROTECTED;

    -- Pcte_activity.lock.internal_mode corresponds to the PCTE datatype
    -- Lock_internal_mode.

    -- 16.2.4 LOCK_RESET_INTERNAL_MODE

    procedure reset_internal_mode (
        object : in     Pcte.object_reference;
        status : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 16.2.5 LOCK_SET_INTERNAL_MODE

    procedure set_internal_mode (
        object    : in     Pcte.object_reference;
        lock_mode : in     Pcte_activity.lock.internal_mode;
        wait_flag : in     Pcte.boolean := TRUE;
        status    : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- 16.2.6 LOCK_SET_OBJECT

    procedure set_object (
        object    : in     Pcte.object_reference;
        lock_mode : in     Pcte_activity.lock.lock_set_mode;
        wait_flag : in     Pcte.boolean;
        scope     : in     Pcte.object_scope;
        status    : in     Pcte_error.handle := EXCEPTION_ONLY);
```

```
        -- 16.2.7 LOCK_UNSET_OBJECT

        procedure unset_object (
            object : in     Pcte.object_reference;
            scope  : in     Pcte.object_scope;
            status : in     Pcte_error.handle := EXCEPTION_ONLY);

    end lock;

end Pcte_activity;
```

# 17 Replication

```
with Pcte, Pcte_error;

package Pcte_replica_set is

    use Pcte_error;
```

## 17.1 Replication datatypes

```
    -- None.
```

## 17.2 Replication operations

```
    -- 17.2.1 REPLICA_SET_ADD_COPY_VOLUME

    procedure add_copy_volume (
        replica_set     : in    Pcte.object_reference;
        copy_volume     : in    Pcte.object_reference;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 17.2.2 REPLICA_SET_CREATE

    procedure create (
        master_volume   : in        Pcte.object_reference;
        identifier      : in        Pcte.natural;
        replica_set     : in out    Pcte.object_reference;
        status          : in        Pcte_error.handle := EXCEPTION_ONLY);

    -- 17.2.3 REPLICA_SET_REMOVE

    procedure remove (
        replica_set     : in    Pcte.object_reference;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 17.2.4 REPLICA_SET_REMOVE_COPY_VOLUME

    procedure remove_copy_volume (
        replica_set     : in    Pcte.object_reference;
        copy_volume     : in    Pcte.object_reference;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
end Pcte_replica_set;

with Pcte, Pcte_error;
package Pcte_replicated_object is
    use Pcte_error;

    -- 17.2.5 REPLICATED_OBJECT_CREATE

    procedure create (
        replica_set    : in    Pcte.object_reference;
        object         : in    Pcte.object_reference;
        status         : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 17.2.6 REPLICATED_OBJECT_DELETE_REPLICA

    procedure delete_replica (
        object         : in    Pcte.object_reference;
        copy_volume    : in    Pcte.object_reference;
        status         : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 17.2.7 REPLICATED_OBJECT_DUPLICATE

    procedure duplicate (
        object         : in    Pcte.object_reference;
        volume         : in    Pcte.object_reference;
        copy_volume    : in    Pcte.object_reference;
        status         : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 17.2.8 REPLICATED_OBJECT_REMOVE

    procedure remove (
        object  : in    Pcte.object_reference;
        status  : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 17.2.9 WORKSTATION_SELECT_REPLICA_VOLUME

    --   See 18.2.

    -- 17.2.10 WORKSTATION_UNSELECT_REPLICA_VOLUME

    --   See 18.2.
end Pcte_replicated_object;
```

## 18 Network connection

```
with Pcte, Pcte_error, Pcte_device, Pcte_discretionary, Pcte_volume;
package Pcte_workstation is
    use Pcte, Pcte_error, Pcte.reference;
```

## 18.1 Network connection datatypes

```
type connection_status is (LOCAL, CLIENT, CONNECTED, AVAILABLE);

-- Pcte_workstation.connection_status corresponds to the PCTE datatype Connection_status.

subtype requested_connection_status is Pcte_workstation.connection_status
    range LOCAL .. CONNECTED;

-- Pcte_workstation.requested_connection_status corresponds to the PCTE datatype
-- Requested_connection_status.

type work_status_item is (ACTIVITY_REMOTE_LOCKS, ACTIVITY_LOCAL_LOCKS,
    TRANSACTION_REMOTE_LOCKS, TRANSACTION_LOCAL_LOCKS,
    QUEUE_REMOTE, QUEUE_LOCAL, RECEIVE_REMOTE, RECEIVE_LOCAL,
    CHILD_REMOTE, CHILD_LOCAL);

-- Pcte_workstation.work_status_item corresponds to the PCTE datatype Work_status_item.

type work_status is array (Pcte_workstation.work_status_item) of Pcte.boolean;

-- Pcte_workstation.work_status corresponds to the PCTE datatype Work_status_item.

type workstation_status is record
    connection    : Pcte_workstation.connection_status;
    work          : Pcte_workstation.work_status;
end record;

-- Pcte_workstation.workstation_status corresponds to the PCTE datatype
-- Workstation_status.
```

## 18.2 Network connection operations

```
-- 18.2.1 WORKSTATION_CONNECT

procedure connect (
    pcte_status   : in    Pcte_workstation.requested_connection_status;
    status        : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 18.2.2 WORKSTATION_CREATE

-- The effect of assigning a Volume_designator to the parameter administration_volume is
-- obtained by the first overloaded procedure. The effect of assigning a
-- New_administration_volume to administration_volume is obtained by the second
-- overloaded procedure.

procedure create (
    execution_site_identifier   : in    Pcte.natural;
    administration_volume       : in    Pcte.object_reference;
    access_mask                 : in    Pcte_discretionary.object.atomic_access_rights;
    node_name                   : in    Pcte.text;
    machine_name                : in    Pcte.text;
    status                      : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
procedure create (
    execution_site_identifier   : in    Pcte.natural;
    foreign_device              : in    Pcte.string;
    administration_volume       : in    Pcte_volume.volume_identifier;
    volume_characteristics      : in    Pcte.string;
    device                      : in    Pcte_device.device_identifier;
    device_characteristics      : in    Pcte.string;
    access_mask                 : in    Pcte_discretionary.object.atomic_access_rights;
    node_name                   : in    Pcte.text;
    machine_name                : in    Pcte.text;
    status                      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 18.2.3 WORKSTATION_DELETE

procedure delete (
    station     : in    Pcte.object_reference;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 18.2.4 WORKSTATION_DISCONNECT

procedure disconnect (
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 18.2.5 WORKSTATION_GET_STATUS

function get_status (
    station     : Pcte.object_reference := LOCAL_WORKSTATION;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte_workstation.workstation_status;

-- 18.2.6 WORKSTATION_REDUCE_CONNECTION

procedure reduce_connection (
    pcte_status     : in    Pcte_workstation.requested_connection_status;
    station         : in    Pcte.object_reference := LOCAL_WORKSTATION;
    force           : in    Pcte.boolean;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 17.2.9 WORKSTATION_SELECT_REPLICA_VOLUME

procedure select_replica_volume (
    station         : in    Pcte.object_reference;
    replica_set     : in    Pcte.object_reference;
    volume          : in    Pcte.object_reference;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 17.2.10 WORKSTATION_UNSELECT_REPLICA_VOLUME

procedure unselect_replica_volume (
    station         : in    Pcte.object_reference;
    replica_set     : in    Pcte.object_reference;
    status          : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
end Pcte_workstation;
```

### 18.3 Foreign system operations

```
-- 18.3.1 CONTENTS_COPY_FROM_FOREIGN_SYSTEM
--   See 12.2.
-- 18.3.2 CONTENTS_COPY_TO_FOREIGN_SYSTEM
--   See 12.2.
```

### 18.4 Time operations

```
with Pcte, Pcte_error;
package Pcte_time is
    use Pcte_error;
    -- 18.4.1 TIME_GET
    function get (
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.calendar.time;
    -- 18.4.2 TIME_SET
    procedure set (
        time    : in    Pcte.calendar.time;
        status  : in    Pcte_error.handle := EXCEPTION_ONLY);
end Pcte_time;
```

## 19 Discretionary security

```
with Pcte, Pcte_error;
package Pcte_discretionary is
    use Pcte_error;
```

### 19.1 Discretionary security datatypes

```
    type group_identifier is new Pcte.natural;
    --  Pcte_discretionary.group_identifier corresponds to the PCTE datatype Group_identifier.
    package object is
        type mode_value is (UNCHANGED, GRANTED, UNDEFINED, DENIED,
            PARTIALLY_DENIED);
        subtype requested_mode_value is Pcte_discretionary.object.mode_value
            range UNCHANGED .. DENIED;
```

```
subtype access_mode_value is Pcte_discretionary.object.mode_value
    range GRANTED .. PARTIALLY_DENIED;

-- Pcte_discretionary.object.access_mode_value corresponds to the PCTE datatype
-- Discretionary_access_mode_value.

subtype atomic_access_mode_value is Pcte_discretionary.object.mode_value
    range GRANTED .. DENIED;

-- Pcte_discretionary.object.atomic_access_mode_value corresponds to the PCTE
-- datatype Atomic_discretionary_access_mode_value.

type access_mode is (APPEND_CONTENTS, APPEND_IMPLICIT, APPEND_LINKS,
    CONTROL_DISCRETIONARY, CONTROL_MANDATORY,
    CONTROL_OBJECT, DELETE, EXECUTE,
    EXPLOIT_CONSUMER_IDENTITY, EXPLOIT_DEVICE, EXPLOIT_SCHEMA,
    NAVIGATE, OWNER, READ_ATTRIBUTES, READ_CONTENTS,
    READ_LINKS, STABILIZE,  WRITE_ATTRIBUTES, WRITE_CONTENTS,
    WRITE_IMPLICIT, WRITE_LINKS);

-- Pcte_discretionary.object.access_mode corresponds to the PCTE datatype
-- Discretionary_access_mode.

type access_modes is array (Pcte_discretionary.object.access_mode) of Pcte.boolean;

-- Pcte_discretionary.object.access_modes corresponds to the PCTE datatype
-- Discretionary_access_modes.

type requested_access_rights is array (Pcte_discretionary.object.access_mode)
    of Pcte_discretionary.object.requested_mode_value;

type access_rights is array (Pcte_discretionary.object.access_mode)
    of Pcte_discretionary.object.access_mode_value;

-- Pcte_discretionary.object.access_rights corresponds to the PCTE datatype
-- Access_rights. The element indexed by a particular Pcte_discretionary.object.
-- access_mode value in an access_rights value corresponds to the image of the
-- corresponding Access_mode value in the corresponding Access_rights map.

type atomic_access_rights is array (Pcte_discretionary.object.access_mode)
    of Pcte_discretionary.object.atomic_access_mode_value;

-- Pcte_discretionary.object.atomic_access_rights corresponds to the PCTE datatype
-- Atomic_access_rights. The element indexed by a particular Pcte_discretionary.object.
-- access_mode value in an atomic_access_rights value corresponds to the image of the
-- corresponding Access_mode value in the corresponding Atomic_access_rights map.

type acl_entry is record
    group       : Pcte_discretionary.group_identifier;
    access_mask : Pcte_discretionary.object.access_rights;
end record;

-- Pcte_discretionary.object.acl_entry corresponds to the maplet type of the PCTE
-- datatype Acl, i.e. the fields of a record of that type correspond to a group identifier and
-- its image Access_rights value in some Acl map value.
```

```
-- The semantics of the operations of this package are defined in 8.2.8.

package acl_entries is

    type sequence is limited private;

    -- Pcte_discretionary.object.acl_entries.sequence corresponds to the PCTE datatype
    -- Acl.

    function get (
        list        : Pcte_discretionary.object.acl_entries.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte_discretionary.object.acl_entry;

    procedure insert (
        list        : in out    Pcte_discretionary.object.acl_entries.sequence;
        item        : in        Pcte_discretionary.object.acl_entry;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure replace (
        list        : in out    Pcte_discretionary.object.acl_entries.sequence;
        item        : in        Pcte_discretionary.object.acl_entry;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure append (
        list        : in out    Pcte_discretionary.object.acl_entries.sequence;
        item        : in        Pcte_discretionary.object.acl_entry;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure delete (
        list        : in out    Pcte_discretionary.object.acl_entries.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure copy (
        into_list   : in out    Pcte_discretionary.object.acl_entries.sequence;
        from_list   : in        Pcte_discretionary.object.acl_entries.sequence;
        into_index  : in        Pcte.natural := Pcte.natural'LAST;
        from_index  : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list        : Pcte_discretionary.object.acl_entries.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.natural;
```

```
        function index_of (
            list        : Pcte_discretionary.object.acl_entries.sequence;
            item        : Pcte_discretionary.object.acl_entry;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return        Pcte.integer;

        function are_equal (
            first       : Pcte_discretionary.object.acl_entries.sequence;
            second      : Pcte_discretionary.object.acl_entries.sequence;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return        Pcte.boolean;

        procedure normalize (
            list        : in out    Pcte_discretionary.object.acl_entries.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);

        procedure discard (
            list        : in out    Pcte_discretionary.object.acl_entries.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    private

        implementation-defined

    end acl_entries;
```

## 19.2 Discretionary access control operations

```
    -- 19.2.1 GROUP_GET_IDENTIFIER

    --   See 19.3.

    -- 19.2.2 OBJECT_CHECK_PERMISSION

    function check_permission (
        object      : Pcte.object_reference;
        modes       : Pcte_discretionary.object.access_modes;
        scope       : Pcte.object_scope;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.boolean;

    -- 19.2.3 OBJECT_GET_ACL

    procedure get_acl (
        object  : in        Pcte.object_reference;
        scope   : in        Pcte.object_scope;
        acl     : in out    Pcte_discretionary.object.acl_entries.sequence;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
-- 19.2.4 OBJECT_SET_ACL_ENTRY

procedure set_acl_entry (
    object : in    Pcte.object_reference;
    group  : in    Pcte_discretionary.group_identifier;
    modes  : in    Pcte_discretionary.object.requested_access_rights;
    scope  : in    Pcte.object_scope;
    status : in    Pcte_error.handle := EXCEPTION_ONLY);

end object;
```

## 19.3 Discretionary security administration operations

```
package group is

    -- 19.2.1 GROUP_GET_IDENTIFIER

    function get_identifier (
        group      : Pcte.object_reference;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte_discretionary.group_identifier;

    -- 19.3.1 GROUP_INITIALIZE

    function initialize (
        group      : Pcte.object_reference;
        status     : Pcte_error.handle := EXCEPTION_ONLY)
        return       Pcte_discretionary.group_identifier;

    -- 19.3.2 GROUP_REMOVE

    procedure remove (
        group  : in    Pcte.object_reference;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 19.3.3 GROUP_RESTORE

    procedure restore (
        group      : in    Pcte.object_reference;
        identifier : in    Pcte_discretionary.group_identifier;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);

end group;


package program_group is

    -- 19.3.4 PROGRAM_GROUP_ADD_MEMBER

    procedure add_member (
        group      : in    Pcte.object_reference;
        program    : in    Pcte.object_reference;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
    -- 19.3.5 PROGRAM_GROUP_ADD_SUBGROUP

    procedure add_subgroup (
        group      : in    Pcte.object_reference;
        subgroup   : in    Pcte.object_reference;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 19.3.6 PROGRAM_GROUP_REMOVE_MEMBER

    procedure remove_member (
        group      : in    Pcte.object_reference;
        program    : in    Pcte.object_reference;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 19.3.7 PROGRAM_GROUP_REMOVE_SUBGROUP

    procedure remove_subgroup (
        group      : in    Pcte.object_reference;
        subgroup   : in    Pcte.object_reference;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);

end program_group;


package user_group is

    -- 19.3.8 USER_GROUP_ADD_MEMBER

    procedure add_member (
        group  : in    Pcte.object_reference;
        user   : in    Pcte.object_reference;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 19.3.9 USER_GROUP_ADD_SUBGROUP

    procedure add_subgroup (
        group      : in    Pcte.object_reference;
        subgroup   : in    Pcte.object_reference;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 19.3.10 USER_GROUP_REMOVE_MEMBER

    procedure remove_member (
        group  : in    Pcte.object_reference;
        user   : in    Pcte.object_reference;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 19.3.11 USER_GROUP_REMOVE_SUBGROUP

    procedure remove_subgroup (
        group      : in    Pcte.object_reference;
        subgroup   : in    Pcte.object_reference;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);

end user_group;
```

```
end Pcte_discretionary;
```

# 20 Mandatory security

```
with Pcte, Pcte_error, Pcte_discretionary;
package Pcte_mandatory is
    use Pcte_error;
```

## 20.1 Mandatory security datatypes

```
    type security_label is new Pcte.string;
    -- Pcte_mandatory.security_label corresponds to the PCTE datatype Security_label.
    type floating_level is (NO_FLOAT, FLOAT_IN, FLOAT_OUT, FLOAT_IN_OUT);
    -- Pcte_mandatory.floating_level corresponds to the PCTE datatype Floating_level.
```

## 20.2 Mandatory security operations

```
    package device is

        -- 20.2.1 DEVICE_SET_CONFIDENTIALITY_RANGE

        procedure set_confidentiality_range (
            device     : in    Pcte.object_reference;
            high_label: in    Pcte_mandatory.security_label;
            low_label : in    Pcte_mandatory.security_label;
            status     : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 20.2.2 DEVICE_SET_INTEGRITY_RANGE

        procedure set_integrity_range (
            device     : in    Pcte.object_reference;
            high_label: in    Pcte_mandatory.security_label;
            low_label : in    Pcte_mandatory.security_label;
            status     : in    Pcte_error.handle := EXCEPTION_ONLY);
    end device;

    package execution_site is

        -- 20.2.3 EXECUTION_SITE_SET_CONFIDENTIALITY_RANGE

        procedure set_confidentiality_range (
            execution_site    : in    Pcte.object_reference;
            high_label        : in    Pcte_mandatory.security_label;
            low_label         : in    Pcte_mandatory.security_label;
            status            : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
    -- 20.2.4 EXECUTION_SITE_SET_INTEGRITY_RANGE

    procedure set_integrity_range (
        execution_site    : in    Pcte.object_reference;
        high_label        : in    Pcte_mandatory.security_label;
        low_label         : in    Pcte_mandatory.security_label;
        status            : in    Pcte_error.handle := EXCEPTION_ONLY);
end execution_site;

package object is

    -- 20.2.5 OBJECT_SET_CONFIDENTIALITY_LABEL

    procedure set_confidentiality_label (
        object : in    Pcte.object_reference;
        label  : in    Pcte_mandatory.security_label;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 20.2.6 OBJECT_SET_INTEGRITY_LABEL

    procedure set_integrity_label (
        object : in    Pcte.object_reference;
        label  : in    Pcte_mandatory.security_label;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);
end object;

package volume is

    -- 20.2.7 VOLUME_SET_CONFIDENTIALITY_RANGE

    procedure set_confidentiality_range (
        volume     : in    Pcte.object_reference;
        high_label : in    Pcte_mandatory.security_label;
        low_label  : in    Pcte_mandatory.security_label;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 20.2.8 VOLUME_SET_INTEGRITY_RANGE

    procedure set_integrity_range (
        volume     : in    Pcte.object_reference;
        high_label : in    Pcte_mandatory.security_label;
        low_label  : in    Pcte_mandatory.security_label;
        status     : in    Pcte_error.handle := EXCEPTION_ONLY);
end volume;
```

## 20.3 Mandatory security administration operations

```
package confidentiality_class is
```

```
    -- 20.3.1 CONFIDENTIALITY_CLASS_INITIALIZE

    procedure initialize (
        object              : in    Pcte.object_reference;
        class_name          : in    Pcte.name;
        to_be_dominated     : in    Pcte.object_reference;
        status              : in    Pcte_error.handle := EXCEPTION_ONLY);

    procedure initialize (
        object          : in    Pcte.object_reference;
        class_name      : in    Pcte.name;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);
end confidentiality_class;

package group is

    -- 20.3.2 GROUP_DISABLE_FOR_CONFIDENTIALITY_DOWNGRADE

    procedure disable_for_confidentiality_downgrade (
        group                   : in    Pcte.object_reference;
        confidentiality_class   : in    Pcte.object_reference;
        status                  : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 20.3.3 GROUP_DISABLE_FOR_INTEGRITY_UPGRADE

    procedure disable_for_integrity_upgrade (
        group           : in    Pcte.object_reference;
        integrity_class : in    Pcte.object_reference;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 20.3.4 GROUP_ENABLE_FOR_CONFIDENTIALITY_DOWNGRADE

    procedure enable_for_confidentiality_downgrade (
        group               : in    Pcte.object_reference;
        confidentiality_class : in  Pcte.object_reference;
        status              : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 20.3.5 GROUP_ENABLE_FOR_INTEGRITY_UPGRADE

    procedure enable_for_integrity_upgrade (
        group           : in    Pcte.object_reference;
        integrity_class : in    Pcte.object_reference;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);
end group;

package integrity_class is
```

```
        -- 20.3.6 INTEGRITY_CLASS_INITIALIZE

        procedure initialize (
            object              : in    Pcte.object_reference;
            class_name          : in    Pcte.name;
            to_be_dominated     : in    Pcte.object_reference;
            status              : in    Pcte_error.handle := EXCEPTION_ONLY);

        procedure initialize (
            object          : in    Pcte.object_reference;
            class_name      : in    Pcte.name;
            status          : in    Pcte_error.handle := EXCEPTION_ONLY);
    end integrity_class;

    package user is

        -- 20.3.7 USER_EXTEND_CONFIDENTIALITY_CLEARANCE

        procedure extend_confidentiality_clearance (
            user                    : in    Pcte.object_reference;
            confidentiality_class   : in    Pcte.object_reference;
            status                  : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 20.3.8 USER_EXTEND_INTEGRITY_CLEARANCE

        procedure extend_integrity_clearance (
            user            : in    Pcte.object_reference;
            integrity_class : in    Pcte.object_reference;
            status          : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 20.3.9 USER_REDUCE_CONFIDENTIALITY_CLEARANCE

        procedure reduce_confidentiality_clearance (
            user                    : in    Pcte.object_reference;
            confidentiality_class   : in    Pcte.object_reference;
            status                  : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 20.3.10 USER_REDUCE_INTEGRITY_CLEARANCE

        procedure reduce_integrity_clearance (
            user            : in    Pcte.object_reference;
            integrity_class : in    Pcte.object_reference;
            status          : in    Pcte_error.handle := EXCEPTION_ONLY);

    end user;
end Pcte_mandatory;
```

## 20.4 Mandatory security operations for processes

```
-- See 13.3.
```

# 21 Auditing

```
with Pcte, Pcte_error, Pcte_discretionary, Pcte_mandatory;

package Pcte_audit is

    use Pcte_error;
```

## 21.1 Auditing datatypes

```
    type event_type is (  WRITE, READ, COPY, ACCESS_CONTENTS, EXPLOIT,
        CHANGE_ACCESS_CONTROL_LIST, CHANGE_LABEL,
        USE_PREDEFINED_GROUP, SET_USER_IDENTITY,
        WRITE_CONFIDENTIALITY_VIOLATION,
        READ_CONFIDENTIALITY_VIOLATION, WRITE_INTEGRITY_VIOLATION,
        READ_INTEGRITY_VIOLATION, COVERT_CHANNEL, INFORMATION,
        CHANGE_IDENTIFICATION, SELECT_AUDIT_EVENT,
        SECURITY_ADMINISTRATION);

    -- Pcte_audit.event_type corresponds to the union of the PCTE datatypes
    -- Selectable_event_type and Mandatory_event_type.

    subtype selectable_event_type is Pcte_audit.event_type range WRITE .. INFORMATION;

    -- Pcte_audit.selectable_event_type corresponds to the PCTE datatype
    -- Selectable_event_type.

    subtype mandatory_event_type is Pcte_audit.event_type
        range CHANGE_IDENTIFICATION .. SECURITY_ADMINISTRATION;

    -- Pcte_audit.mandatory_event_type corresponds to the PCTE datatype
    -- Mandatory_event_type.

    type selected_return_code is (FAILURE, SUCCESS, ANY_CODE);

    -- Pcte_audit.selected_return_code corresponds to the PCTE datatype
    -- Selected_return_code.

    subtype return_code is Pcte_audit.selected_return_code range FAILURE .. SUCCESS;

    -- Pcte_audit.return_code corresponds to the PCTE datatype Return_code.

    type audit_status is (ENABLED, DISABLED);

    -- Pcte_audit.audit_status corresponds to the PCTE datatype Audit_status.

    type general_criterion is record
        selectable_event_type    : Pcte_audit.selectable_event_type;
        return_code              : Pcte_audit.selected_return_code;
    end record;

    -- Pcte_audit.general_criterion corresponds to the PCTE datatype General_criterion.


    -- The semantics of the operations of this package are defined in 8.2.8.

    package general_criteria is
```

```
type sequence is limited private;

-- Pcte_audit.general_criteria.sequence corresponds to the PCTE datatype
-- General_criteria.

function get (
    list        : Pcte_audit.general_criteria.sequence;
    index       : Pcte.natural := Pcte.natural'FIRST;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte_audit.general_criterion;

procedure insert (
    list        : in out    Pcte_audit.general_criteria.sequence;
    item        : in        Pcte_audit.general_criterion;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure replace (
    list        : in out    Pcte_audit.general_criteria.sequence;
    item        : in        Pcte_audit.general_criterion;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure append (
    list        : in out    Pcte_audit.general_criteria.sequence;
    item        : in        Pcte_audit.general_criterion;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure delete (
    list        : in out    Pcte_audit.general_criteria.sequence;
    index       : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure copy (
    into_list   : in out    Pcte_audit.general_criteria.sequence;
    from_list   : in        Pcte_audit.general_criteria.sequence;
    into_index  : in        Pcte.natural := Pcte.natural'LAST;
    from_index  : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

function length_of (
    list        : Pcte_audit.general_criteria.sequence;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.natural;

function index_of (
    list        : Pcte_audit.general_criteria.sequence;
    item        : Pcte_audit.general_criterion;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.integer;
```

```
    function are_equal (
        first       : Pcte_audit.general_criteria.sequence;
        second      : Pcte_audit.general_criteria.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte_audit.general_criteria.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte_audit.general_criteria.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end general_criteria;


type confidentiality_criterion
    (label_length : Pcte.string_length := 0)
is record
    selectable_event_type   : Pcte_audit.selectable_event_type;
    security_label          : Pcte_mandatory.security_label(1..label_length);
end record;

-- Pcte_audit.confidentiality_criterion corresponds to the PCTE datatype
-- Confidentiality_criterion.


-- The semantics of the operations of this package are defined in 8.2.8.

package confidentiality_criteria is

    type sequence is limited private;

    -- Pcte_audit.confidentiality_criteria.sequence corresponds to the
    -- PCTE datatype Confidentiality_criteria.

    function get (
        list        : Pcte_audit.confidentiality_criteria.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte_audit.confidentiality_criterion;

    procedure insert (
        list        : in out    Pcte_audit.confidentiality_criteria.sequence;
        item        : in        Pcte_audit.confidentiality_criterion;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure replace (
        list        : in out    Pcte_audit.confidentiality_criteria.sequence;
        item        : in        Pcte_audit.confidentiality_criterion;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure append (
        list        : in out    Pcte_audit.confidentiality_criteria.sequence;
        item        : in        Pcte_audit.confidentiality_criterion;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure delete (
        list        : in out    Pcte_audit.confidentiality_criteria.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure copy (
        into_list       : in out    Pcte_audit.confidentiality_criteria.sequence;
        from_list       : in        Pcte_audit.confidentiality_criteria.sequence;
        into_index      : in        Pcte.natural := Pcte.natural'LAST;
        from_index      : in        Pcte.natural := Pcte.natural'FIRST;
        count           : in        Pcte.positive := Pcte.positive'LAST;
        status          : in        Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list        : Pcte_audit.confidentiality_criteria.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.natural;

    function index_of (
        list        : Pcte_audit.confidentiality_criteria.sequence;
        item        : Pcte_audit.confidentiality_criterion;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.integer;

    function are_equal (
        first       : Pcte_audit.confidentiality_criteria.sequence;
        second      : Pcte_audit.confidentiality_criteria.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte_audit.confidentiality_criteria.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte_audit.confidentiality_criteria.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined
```

```
end confidentiality_criteria;


type integrity_criterion
    (label_length : Pcte.string_length := 0)
is record
    selectable_event_type   : Pcte_audit.selectable_event_type;
    security_label          : Pcte_mandatory.security_label(1..label_length);
end record;

-- Pcte_audit.integrity_criterion corresponds to the PCTE datatype Integrity_criterion.


-- The semantics of the operations of this package are defined in 8.2.8.

package integrity_criteria is

    type sequence is limited private;

    -- Pcte_audit.integrity_criteria.sequence corresponds to the PCTE datatype
    -- Integrity_criteria.

    function get (
        list        : Pcte_audit.integrity_criteria.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte_audit.integrity_criterion;

    procedure insert (
        list        : in out    Pcte_audit.integrity_criteria.sequence;
        item        : in        Pcte_audit.integrity_criterion;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure replace (
        list        : in out    Pcte_audit.integrity_criteria.sequence;
        item        : in        Pcte_audit.integrity_criterion;
        index       : in        Pcte.natural := Pcte.natural'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure append (
        list        : in out    Pcte_audit.integrity_criteria.sequence;
        item        : in        Pcte_audit.integrity_criterion;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure delete (
        list        : in out    Pcte_audit.integrity_criteria.sequence;
        index       : in        Pcte.natural := Pcte.natural'FIRST;
        count       : in        Pcte.positive := Pcte.positive'LAST;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure copy (
        into_list     : in out   Pcte_audit.integrity_criteria.sequence;
        from_list     : in       Pcte_audit.integrity_criteria.sequence;
        into_index    : in       Pcte.natural := Pcte.natural'LAST;
        from_index    : in       Pcte.natural := Pcte.natural'FIRST;
        count         : in       Pcte.positive := Pcte.positive'LAST;
        status        : in       Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list          : Pcte_audit.integrity_criteria.sequence;
        status        : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.natural;

    function index_of (
        list          : Pcte_audit.integrity_criteria.sequence;
        item          : Pcte_audit.integrity_criterion;
        status        : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.integer;

    function are_equal (
        first         : Pcte_audit.integrity_criteria.sequence;
        second        : Pcte_audit.integrity_criteria.sequence;
        status        : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.boolean;

    procedure normalize (
        list          : in out   Pcte_audit.integrity_criteria.sequence;
        status        : in       Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list          : in out   Pcte_audit.integrity_criteria.sequence;
        status        : in       Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end integrity_criteria;


type object_criterion is record
    selectable_event_type    : Pcte_audit.selectable_event_type;
    object_reference         : Pcte.object_reference;
end record;

-- Pcte_audit.object_criterion corresponds to the PCTE datatype Object_criterion.

-- The semantics of the operations of this package are defined in 8.2.8.

package object_criteria is

    type sequence is limited private;
```

```
-- Pcte_audit.object_criteria.sequence corresponds to the
-- PCTE datatype Object_criteria.

function get (
    list        : Pcte_audit.object_criteria.sequence;
    index       : Pcte.natural := Pcte.natural'FIRST;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte_audit.object_criterion;

procedure insert (
    list        : in out    Pcte_audit.object_criteria.sequence;
    item        : in        Pcte_audit.object_criterion;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure replace (
    list        : in out    Pcte_audit.object_criteria.sequence;
    item        : in        Pcte_audit.object_criterion;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure append (
    list        : in out    Pcte_audit.object_criteria.sequence;
    item        : in        Pcte_audit.object_criterion;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure delete (
    list        : in out    Pcte_audit.object_criteria.sequence;
    index       : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure copy (
    into_list   : in out    Pcte_audit.object_criteria.sequence;
    from_list   : in        Pcte_audit.object_criteria.sequence;
    into_index  : in        Pcte.natural := Pcte.natural'LAST;
    from_index  : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

function length_of (
    list        : Pcte_audit.object_criteria.sequence;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.natural;

function index_of (
    list        : Pcte_audit.object_criteria.sequence;
    item        : Pcte_audit.object_criterion;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.integer;
```

```
    function are_equal (
        first       : Pcte_audit.object_criteria.sequence;
        second      : Pcte_audit.object_criteria.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte_audit.object_criteria.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte_audit.object_criteria.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
private
    implementation-defined
end object_criteria;


type user_criterion is record
    selectable_event_type   : Pcte_audit.selectable_event_type;
    group                   : Pcte_discretionary.group_identifier;
end record;
-- Pcte_audit.user_criterion corresponds to the PCTE datatype User_criterion.


-- The semantics of the operations of this package are defined in 8.2.8.
package user_criteria is

    type sequence is limited private;

    -- Pcte_audit.user_criteria.sequence corresponds to the PCTE datatype User_criteria.

    function get (
        list        : Pcte_audit.user_criteria.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte_audit.user_criterion;

    procedure insert (
        list    : in out    Pcte_audit.user_criteria.sequence;
        item    : in        Pcte_audit.user_criterion;
        index   : in        Pcte.natural := Pcte.natural'LAST;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure replace (
        list    : in out    Pcte_audit.user_criteria.sequence;
        item    : in        Pcte_audit.user_criterion;
        index   : in        Pcte.natural := Pcte.natural'LAST;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure append (
        list    : in out    Pcte_audit.user_criteria.sequence;
        item    : in        Pcte_audit.user_criterion;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure delete (
        list    : in out    Pcte_audit.user_criteria.sequence;
        index   : in        Pcte.natural := Pcte.natural'FIRST;
        count   : in        Pcte.positive := Pcte.positive'LAST;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure copy (
        into_list       : in out    Pcte_audit.user_criteria.sequence;
        from_list       : in        Pcte_audit.user_criteria.sequence;
        into_index      : in        Pcte.natural := Pcte.natural'LAST;
        from_index      : in        Pcte.natural := Pcte.natural'FIRST;
        count           : in        Pcte.positive := Pcte.positive'LAST;
        status          : in        Pcte_error.handle := EXCEPTION_ONLY);

    function length_of (
        list        : Pcte_audit.user_criteria.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.natural;

    function index_of (
        list        : Pcte_audit.user_criteria.sequence;
        item        : Pcte_audit.user_criterion;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.integer;

    function are_equal (
        first       : Pcte_audit.user_criteria.sequence;
        second      : Pcte_audit.user_criteria.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.boolean;

    procedure normalize (
        list    : in out    Pcte_audit.user_criteria.sequence;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list    : in out    Pcte_audit.user_criteria.sequence;
        status  : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end user_criteria;
```

## 21.2 Auditing operations

```
-- 21.2.1 AUDIT_ADD_CRITERION

-- AUDIT_ADD_CRITERION is mapped to the overloaded procedures
-- Pcte_audit.add_criterion according to the type of the parameter criterion.

procedure add_criterion (
    station    : in    Pcte.object_reference;
    criterion  : in    Pcte_audit.general_criterion;
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure add_criterion (
    station    : in    Pcte.object_reference;
    criterion  : in    Pcte_audit.confidentiality_criterion;
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure add_criterion (
    station    : in    Pcte.object_reference;
    criterion  : in    Pcte_audit.integrity_criterion;
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure add_criterion (
    station    : in    Pcte.object_reference;
    criterion  : in    Pcte_audit.object_criterion;
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure add_criterion (
    station    : in    Pcte.object_reference;
    criterion  : in    Pcte_audit.user_criterion;
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 21.2.2 AUDIT_FILE_COPY_AND_RESET

-- See below.

-- 21.2.3 AUDIT_FILE_READ

-- See below.

-- 21.2.4 AUDIT_GET_CRITERIA

-- AUDIT_GET_CRITERIA is mapped to the overloaded procedures Pcte_audit.get_criteria
-- according to the type of the result criteria. The parameter criterion_type which determines
-- the type of the result is omitted.

procedure get_criteria (
    station    : in        Pcte.object_reference;
    criteria   : in out    Pcte_audit.general_criteria.sequence;
    status     : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
procedure get_criteria (
    station     : in        Pcte.object_reference;
    criteria    : in out    Pcte_audit.confidentiality_criteria.sequence;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure get_criteria (
    station     : in        Pcte.object_reference;
    criteria    : in out    Pcte_audit.integrity_criteria.sequence;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure get_criteria (
    station     : in        Pcte.object_reference;
    criteria    : in out    Pcte_audit.object_criteria.sequence;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure get_criteria (
    station     : in        Pcte.object_reference;
    criteria    : in out    Pcte_audit.user_criteria.sequence;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

-- 21.2.5 AUDIT_RECORD_WRITE

--  See below.

-- 21.2.6 AUDIT_REMOVE_CRITERION

--  AUDIT_REMOVE_CRITERION is mapped to the overloaded procedures
--  Pcte_audit.remove_criterion according to the type of the parameter criterion.

procedure remove_criterion (
    station     : in    Pcte.object_reference;
    criterion   : in    Pcte_audit.selectable_event_type;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure remove_criterion (
    station     : in    Pcte.object_reference;
    criterion   : in    Pcte_audit.confidentiality_criterion;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure remove_criterion (
    station     : in    Pcte.object_reference;
    criterion   : in    Pcte_audit.integrity_criterion;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

procedure remove_criterion (
    station     : in    Pcte.object_reference;
    criterion   : in    Pcte_audit.object_criterion;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
procedure remove_criterion (
    station     : in    Pcte.object_reference;
    criterion   : in    Pcte_audit.user_criterion;
    status      : in    Pcte_error.handle := EXCEPTION_ONLY);

package file is

    type auditing_record (
        event_type      : Pcte_audit.event_type := INFORMATION;
        id_1_length     : Pcte.string_length := 0;
        id_2_length     : Pcte.string_length := 0;
        id_3_length     : Pcte.string_length := 0;
        id_4_length     : Pcte.string_length := 0;
        text_length     : Pcte.string_length := 0)
    is record
        user            : Pcte_discretionary.group_identifier;
        time            : Pcte.calendar.time;
        workstation     : Pcte.exact_identifier (1 .. id_1_length);
        return_code     : Pcte_audit.return_code;
        process         : Pcte.exact_identifier (1 .. id_2_length);
        case event_type is
            when EXPLOIT =>
                new_process       : Pcte.exact_identifier (1 .. id_3_length);
                exploited_object  : Pcte.exact_identifier (1 .. id_4_length);
            when INFORMATION =>
                text    : Pcte.string (1..text_length);
            when COPY | CHANGE_IDENTIFICATION =>
                source          : Pcte.exact_identifier (1 .. id_3_length);
                destination     : Pcte.exact_identifier (1 .. id_4_length);
            when USE_PREDEFINED_GROUP | SET_USER_IDENTITY =>
                group           : Pcte.exact_identifier (1 .. id_3_length);
            when others =>
                object      : Pcte.exact_identifier (1 .. id_3_length);
        end case;
    end record;
    -- Pcte_audit.file.auditing_record corresponds to the PCTE datatype Auditing_record,
    -- which is a union type.  Each of its component datatypes is mapped to the particular
    -- constrained record type which has the discriminant event_type set to the
    -- Pcte_audit.event_type value corresponding to that component.


-- The semantics of the operations of this package are defined in 8.2.8.

package auditing_records is

    type sequence is limited private;

    -- Pcte_audit.file.auditing_records.sequence corresponds to the PCTE datatype
    -- Audit_file.
```

```
function get (
    list        : Pcte_audit.file.auditing_records.sequence;
    index       : Pcte.natural := Pcte.natural'FIRST;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte_audit.file.auditing_record;

procedure insert (
    list        : in out    Pcte_audit.file.auditing_records.sequence;
    item        : in        Pcte_audit.file.auditing_record;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure replace (
    list        : in out    Pcte_audit.file.auditing_records.sequence;
    item        : in        Pcte_audit.file.auditing_record;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure append (
    list        : in out    Pcte_audit.file.auditing_records.sequence;
    item        : in        Pcte_audit.file.auditing_record;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure delete (
    list        : in out    Pcte_audit.file.auditing_records.sequence;
    index       : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure copy (
    into_list   : in out    Pcte_audit.file.auditing_records.sequence;
    from_list   : in        Pcte_audit.file.auditing_records.sequence;
    into_index  : in        Pcte.natural := Pcte.natural'LAST;
    from_index  : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

function length_of (
    list        : Pcte_audit.file.auditing_records.sequence;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.natural;

function index_of (
    list        : Pcte_audit.file.auditing_records.sequence;
    item        : Pcte_audit.file.auditing_record;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.integer;
```

```
    function are_equal (
        first       : Pcte_audit.file.auditing_records.sequence;
        second      : Pcte_audit.file.auditing_records.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte_audit.file.auditing_records.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte_audit.file.auditing_records.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);
private
    implementation-defined
end auditing_records;


    -- 21.2.2 AUDIT_FILE_COPY_AND_RESET

    procedure copy_and_reset (
        source          : in    Pcte.object_reference;
        destination     : in    Pcte.object_reference;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 21.2.3 AUDIT_FILE_READ

    procedure read (
        audit_file          : in        Pcte.object_reference;
        no_of_records       : in        Pcte.natural;
        records             : in out    Pcte_audit.file.auditing_records.sequence;
        at_beginning        : in        Pcte.boolean := FALSE;
        status              : in        Pcte_error.handle := EXCEPTION_ONLY);

    --  Pcte_audit.file.read reads no_of_records records or to the end of the audit file, the
    --  whichever is fewer, from the current position in the audit file for the calling process.
    --  The current position is incremented by the number of records read.  If the parameter
    --  at_beginning is set to TRUE, the current position in audit_file is first set to the
    --  beginning of the audit file before reading.

    -- 21.2.5 AUDIT_RECORD_WRITE

    procedure write (
        text    : in    Pcte.string;
        status  : in    Pcte_error.handle := EXCEPTION_ONLY);
end file;
```

```
        -- 21.2.7 AUDIT_SELECTION_CLEAR

        procedure clear_selection (
            station : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 21.2.8 AUDIT_SWITCH_OFF_SELECTION

        procedure switch_off_selection (
            station : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 21.2.9 AUDIT_SWITCH_ON_SELECTION

        procedure switch_on_selection (
            station : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 21.2.10 AUDITING_STATUS

        function get_status (
            station     : Pcte.object_reference;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte_audit.audit_status;

    end Pcte_audit;
```

# 22 Accounting

```
    with Pcte, Pcte_error, Pcte_discretionary;

    package Pcte_accounting is

        use Pcte_error;
```

## 22.1 Accounting datatypes

```
        type consumer_identifier is new Pcte.natural;

        -- Pcte_accounting.consumer_identifier corresponds to the PCTE datatype
        -- Consumer_identifier.

        type resource_identifier is new Pcte.natural;

        -- Pcte_accounting.resource_identifier corresponds to the PCTE datatype
        -- Resource_identifier.

        package log is

            type operation is (SEND, RECEIVE);

            type resource_kind is (WORKSTATION, STATIC_CONTEXT, SDS, DEVICE, FILE,
                PIPE, MESSAGE_QUEUE, INFORMATION);
```

```
type accounting_record (
    kind                    : Pcte_accounting.log.resource_kind := INFORMATION;
    consumer_group_length   : Pcte.string_length := 0;
    resource_group_length   : Pcte.string_length := 0;
    information_length      : Pcte.string_length := 0)
is record
    security_user           : Pcte_discretionary.group_identifier;
    adopted_user_group      : Pcte_discretionary.group_identifier;
    consumer_group          : Pcte.exact_identifier(1..consumer_group_length);
    resource_group          : Pcte.exact_identifier(1..resource_group_length);
    start_time              : Pcte.calendar.time;
    duration                : Pcte.float;
    case kind is
        when WORKSTATION | STATIC_CONTEXT =>
            cpu_time        : Pcte.float;
            system_time     : Pcte.float;
        when FILE | PIPE | DEVICE =>
            read_count      : Pcte.natural;
            write_count     : Pcte.natural;
            read_size       : Pcte.natural;
            write_size      : Pcte.natural;
        when MESSAGE_QUEUE =>
            operation       : Pcte_accounting.log.operation;
            message_size    : Pcte.natural;
        when INFORMATION =>
            information     : Pcte.string(1..information_length);
        when SDS =>
            null;
    end case;
end record;

-- Pcte_accounting.log.accounting_record corresponds to the PCTE datatype
-- Accounting_record, which is a union type.  Each of its component datatypes is
-- mapped to the particular constrained record type which has the discriminant kind set to
-- the Pcte_accounting.log.resource_kind value corresponding to that component.


-- The semantics of the operations of this package are defined in 8.2.8.

package accounting_records is

    type sequence is limited private;

    -- Pcte_accounting.log.accounting_records.sequence corresponds to the PCTE
    -- datatype Accounting_log.
```

```
function get (
    list        : Pcte_accounting.log.accounting_records.sequence;
    index       : Pcte.natural := Pcte.natural'FIRST;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte_accounting.log.accounting_record;

procedure insert (
    list        : in out    Pcte_accounting.log.accounting_records.sequence;
    item        : in        Pcte_accounting.log.accounting_record;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure replace (
    list        : in out    Pcte_accounting.log.accounting_records.sequence;
    item        : in        Pcte_accounting.log.accounting_record;
    index       : in        Pcte.natural := Pcte.natural'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure append (
    list        : in out    Pcte_accounting.log.accounting_records.sequence;
    item        : in        Pcte_accounting.log.accounting_record;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure delete (
    list        : in out    Pcte_accounting.log.accounting_records.sequence;
    index       : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

procedure copy (
    into_list   : in out    Pcte_accounting.log.accounting_records.sequence;
    from_list   : in        Pcte_accounting.log.accounting_records.sequence;
    into_index  : in        Pcte.natural := Pcte.natural'LAST;
    from_index  : in        Pcte.natural := Pcte.natural'FIRST;
    count       : in        Pcte.positive := Pcte.positive'LAST;
    status      : in        Pcte_error.handle := EXCEPTION_ONLY);

function length_of (
    list        : Pcte_accounting.log.accounting_records.sequence;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.natural;

function index_of (
    list        : Pcte_accounting.log.accounting_records.sequence;
    item        : Pcte_accounting.log.accounting_record;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return        Pcte.integer;
```

```
    function are_equal (
        first       : Pcte_accounting.log.accounting_records.sequence;
        second      : Pcte_accounting.log.accounting_records.sequence;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return        Pcte.boolean;

    procedure normalize (
        list        : in out    Pcte_accounting.log.accounting_records.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

    procedure discard (
        list        : in out    Pcte_accounting.log.accounting_records.sequence;
        status      : in        Pcte_error.handle := EXCEPTION_ONLY);

private

    implementation-defined

end accounting_records;
```

## 22.2 Accounting operations

```
-- 22.2.1 ACCOUNTING_LOG_COPY_AND_RESET

procedure copy_and_reset (
    source_log          : in    Pcte.object_reference;
    destination_log     : in    Pcte.object_reference;
    status              : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 22.2.2 ACCOUNTING_LOG_READ

procedure read (
    log                 : in        Pcte.object_reference;
    no_of_records       : in        Pcte.natural;
    records             : in out    Pcte_accounting.log.accounting_records.sequence;
    at_beginning        : in        Pcte.boolean := FALSE;
    status              : in        Pcte_error.handle := EXCEPTION_ONLY);

--  Pcte_accounting.log.read reads no_of_records records or to the end of the accounting
--  log, whichever is fewer, from the current position in the accounting log for the calling
--  process. The current position is incremented by the number of records read. If the
--  parameter at_beginning is set to TRUE, the current position in log is first set to the
--  beginning of the accounting log before reading.

-- 22.2.3 ACCOUNTING_OFF

--  See below.

-- 22.2.4 ACCOUNTING_ON

--  See below.
```

```
    -- 22.2.5 ACCOUNTING_RECORD_WRITE

    procedure write (
        log             : in    Pcte.object_reference;
        information     : in    Pcte.string;
        status          : in    Pcte_error.handle := EXCEPTION_ONLY);
end log;

-- 22.2.3 ACCOUNTING_OFF

procedure off (
    station : in    Pcte.object_reference;
    status  : in    Pcte_error.handle := EXCEPTION_ONLY);

-- 22.2.4 ACCOUNTING_ON

procedure on (
    log     : in    Pcte.object_reference;
    station : in    Pcte.object_reference;
    status  : in    Pcte_error.handle := EXCEPTION_ONLY);

package consumer_group is

    -- 22.2.6 CONSUMER_GROUP_INITIALIZE

    function initialize (
        group       : Pcte.object_reference;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte_accounting.consumer_identifier;

    -- 22.2.7 CONSUMER_GROUP_REMOVE

    procedure remove (
        group  : in    Pcte.object_reference;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);
end consumer_group;

package resource_group is

    -- 22.2.8 RESOURCE_GROUP_ADD_OBJECT

    procedure add_object (
        object : in    Pcte.object_reference;
        group  : in    Pcte.object_reference;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);

    -- 22.2.9 RESOURCE_GROUP_REMOVE

    procedure remove (
        group  : in    Pcte.object_reference;
        status : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
        -- 22.2.10 RESOURCE_GROUP_INITIALIZE

        function initialize (
            group       : Pcte.object_reference;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte_accounting.resource_identifier;

        -- 22.2.11 RESOURCE_GROUP_REMOVE_OBJECT

        procedure remove_object (
            object : in     Pcte.object_reference;
            group  : in     Pcte.object_reference;
            status : in     Pcte_error.handle := EXCEPTION_ONLY);

    end resource_group;

end Pcte_accounting;
```

### 22.3 Consumer identity operations

-- See 13.3.

## 23 References

-- See 8.4.

## 24 Limits

```
with Pcte, Pcte_error;

package Pcte_limit is

    use Pcte_error;
```

```
type limit is (MAX_ACCESS_CONTROL_LIST_LENGTH,
    MAX_ACCOUNT_DURATION, DELTA_ACCOUNT_DURATION,
    MAX_ACCOUNT_INFORMATION_LENGTH,
    MAX_ACTIVITIES,
    MAX_ACTIVITIES_PER_PROCESS,
    MAX_AUDIT_INFORMATION_LENGTH,
    MAX_DIGIT_FLOAT_ATTRIBUTE,
    MAX_FILE_SIZE,
    MAX_FLOAT_ATTRIBUTE, MIN_FLOAT_ATTRIBUTE,
    MAX_INTEGER_ATTRIBUTE, MIN_INTEGER_ATTRIBUTE,
    MAX_KEY_SIZE,
    MAX_KEY_VALUE,
    MAX_LINK_REFERENCE_SIZE,
    MAX_MESSAGE_QUEUE_SPACE,
    MAX_MESSAGE_SIZE,
    MAX_MOUNTED_VOLUMES,
    MAX_NAME_SIZE,
    MAX_NATURAL_ATTRIBUTE,
    MAX_OPEN_OBJECTS,
    MAX_OPEN_OBJECTS_PER_PROCESS,
    MAX_PIPE_SIZE,
    MAX_PRIORITY_VALUE,
    MAX_PROCESSES,
    MAX_PROCESSES_PER_USER,
    MAX_SDS_IN_WORKING_SCHEMA,
    MAX_SECURITY_GROUPS,
    MAX_STRING_ATTRIBUTE_SIZE,
    MAX_TIME_ATTRIBUTE, MIN_TIME_ATTRIBUTE,
    SMALLEST_FLOAT_ATTRIBUTE);

type category is (STANDARD_LIMIT, IMPLEMENTATION_LIMIT,
    REMAINING_LIMIT);

-- Pcte_limit.category indicates to the procedure get_value the category of limit value
-- required. STANDARD_LIMIT indicates the limit bounds given in ISO/IEC 13719-1,
-- clause 24; IMPLEMENTATION_LIMIT indicates the limits imposed by the
-- implementation; and REMAINING_LIMIT indicates the current available quantity of the
-- resource concerned.

procedure get_value (
    value      : in   Pcte_limit.limit;
    result     : out  Pcte.integer;
    unlimited  : out  Pcte.boolean;
    category   : in   Pcte_limit.category := STANDARD_LIMIT;
    status     : in   Pcte_error.handle := EXCEPTION_ONLY);
```

```
    procedure get_value (
        value      : in     Pcte_limit.limit;
        result     : out    Pcte.float;
        unlimited  : out    Pcte.boolean;
        category   : in     Pcte_limit.category := STANDARD_LIMIT;
        status     : in     Pcte_error.handle := EXCEPTION_ONLY);

    procedure get_value (
        value      : in     Pcte_limit.limit;
        result     : out    Pcte.calendar.time;
        unlimited  : out    Pcte.boolean;
        category   : in     Pcte_limit.category := STANDARD_LIMIT;
        status     : in     Pcte_error.handle := EXCEPTION_ONLY);

    procedure get_value (
        value      : in     Pcte_limit.limit;
        result     : out    STANDARD.DURATION;
        unlimited  : out    Pcte.boolean;
        category   : in     Pcte_limit.category := STANDARD_LIMIT;
        status     : in     Pcte_error.handle := EXCEPTION_ONLY);

    -- Pcte_limit.get_value returns the value of the limit value according to the value of
    -- category. The second overloaded procedure is used for MAX_FLOAT_ATTRIBUTE,
    -- MIN_FLOAT_ATTRIBUTE, and SMALLEST_FLOAT_ATTRIBUTE; the third
    -- overloaded procedure is used for MAX_TIME_ATTRIBUTE and
    -- MIN_TIME_ATTRIBUTE; the fourth overloaded procedure is used for
    -- MAX_ACCOUNT_DURATION and DELTA_ACCOUNT_DURATION; and the first
    -- overloaded procedure is used for the rest.

end Pcte_limit;
```

## 25 Errors

```
package Pcte_error is

    type handle is limited private;

    -- Pcte_error.handle is the type of the parameter status of mode in which most subprograms
    -- have. An error handle has two associated properties, called record and raise. If record is
    -- set and an error condition occurs, the enumeration value indicating the error condition is
    -- recorded in the error handle and may be interrogated. If raise is set and an error condition
    -- occurs, an exception is raised. Both record and raise may be set. Error handles are
    -- initialized to EXCEPTION_ONLY.

    EXCEPTION_ONLY : constant Pcte_error.handle;

    -- EXCEPTION_ONLY is a value of an error handle which has raise set, record unset, and
    -- has recorded error condition NO_ERROR.
```

```
---------------------------------        PCTE exceptions        ---------------------------------
  -- The error conditions defined in ISO/IEC 13719-1, annex C, and error conditions defined
  -- in this binding, are divided into a number of sets, each corresponding to an exception.
  -- The exception which may be raised by each error condition is shown by a code letter
  -- against the error condition name in the list below.  The exceptions, their code letters, and
  -- descriptions of the general nature of the corresponding error conditions are as follows.

  ERROR_WITH_USAGE : exception;                                          -- U

  -- indicates that the intended usage of PCTE is an error.

  ERROR_WITH_STATE : exception;                                          -- S

  -- indicates that the call is an error in the context of the state of the object base.

  ERROR_WITH_IDENTIFICATION : exception;                                 -- I

  -- indicates that the parameters fail to identify a valid entity of the intended kind.

  ERROR_WITH_PARAMETER : exception;                                      -- P

  -- indicates a fundamental error in the values of one or more parameters, such that the call
  -- could not be valid under any circumstances

  ERROR_WITH_DEVICE : exception;                                         -- D

  -- indicates that there is a problem with the use of a device or a volume.

  ERROR_WITH_NETWORK : exception;                                        -- N

  -- indicates that there is a problem with access over the network.

  ERROR_FROM_INTERRUPTED : exception;                                    -- R

  -- indicates that the operation has been interrupted.

  ERROR_FROM_TIMEOUT : exception;                                        -- T

  -- indicates that the operation has been timed out.

  ERROR_WITH_TIME : exception;

  -- indicates that there is a problem in the use of the operations "time_of", "+" or "-" of the
  -- package Pcte.calendar for the same reasons as defined by TIME_ERROR in the package
  -- CALENDAR.

---------------------------------        PCTE ERROR CODES        ---------------------------------
type error_code is (

NO_ERROR,

-- Errors defined in ISO/IEC 13719-1:
    Corresponding Exceptions:

ACCESS_CONTROL_WOULD_NOT_BE_GRANTED,                                     -- U
ACCESS_MODE_IS_INCOMPATIBLE,                                             -- P
ACCESS_MODE_IS_NOT_ALLOWED,                                              -- S
```

```
ACCOUNTING_LOG_IS_NOT_ACTIVE,                                          -- U
ACTIVITY_IS_OPERATING_ON_A_RESOURCE,                                   -- S
ACTIVITY_STATUS_IS_INVALID,                                            -- S
ACTIVITY_WAS_NOT_STARTED_BY_CALLING_PROCESS,                           -- U
ARCHIVE_EXISTS,                                                        -- U
ARCHIVE_HAS_ARCHIVED_OBJECTS,                                          -- U
ARCHIVE_IS_INVALID_ON_DEVICE,                                          -- U
ARCHIVE_IS_UNKNOWN,                                                    -- S
ATOMIC_ACL_IS_INCOMPATIBLE_WITH_OWNER_CHANGE,                          -- S
ATTRIBUTE_TYPE_IS_NOT_VISIBLE,                                         -- I
ATTRIBUTE_TYPE_OF_LINK_TYPE_IS_NOT_APPLIED,                            -- U
ATTRIBUTE_TYPE_OF_OBJECT_TYPE_IS_NOT_APPLIED,                          -- U
AUDIT_FILE_IS_NOT_ACTIVE,                                              -- U
BREAKPOINT_IS_NOT_DEFINED,                                             -- I
CARDINALITY_IS_INVALID,                                                -- U
CATEGORY_IS_BAD,                                                       -- P
CLASS_NAME_IS_INVALID,                                                 -- P
CONFIDENTIALITY_CONFINEMENT_WOULD_BE_VIOLATED,                         -- S
CONFIDENTIALITY_CRITERION_IS_NOT_SELECTED,                             -- U
CONFIDENTIALITY_LABEL_IS_INVALID,                                      -- P
CONFIDENTIALITY_WOULD_BE_VIOLATED,                                     -- S
CONNECTION_IS_DENIED,                                                  -- N
CONSUMER_GROUP_IS_IN_USE,                                              -- S
CONSUMER_GROUP_IS_KNOWN,                                               -- S
CONSUMER_GROUP_IS_UNKNOWN,                                             -- I
CONTENTS_IS_NOT_EMPTY,                                                 -- S
CONTENTS_IS_NOT_FILE_CONTENTS,                                         -- U
CONTENTS_IS_NOT_OPEN,                                                  -- S
CONTENTS_OPERATION_IS_INVALID,                                         -- S
CONTROL_WOULD_NOT_BE_GRANTED,                                          -- U
DATA_ARE_NOT_AVAILABLE,                                                -- S
DEFAULT_ACL_WOULD_BE_INCONSISTENT_WITH_DEFAULT_OBJECT_OWNER,           -- U
DEFAULT_ACL_WOULD_BE_INVALID,                                          -- U
DEFINITION_MODE_VALUE_WOULD_BE_INVALID,                                -- U
DESTINATION_OBJECT_TYPE_IS_INVALID,                                    -- I
DEVICE_CHARACTERISTICS_ARE_INVALID,                                    -- D
DEVICE_CONTROL_OPERATION_IS_INVALID,                                   -- D
DEVICE_EXISTS,                                                         -- U
DEVICE_IS_BUSY,                                                        -- D
DEVICE_IS_IN_USE,                                                      -- D
DEVICE_IS_UNKNOWN,                                                     -- I
DEVICE_LIMIT_WOULD_BE_EXCEEDED,                                        -- D
DEVICE_SPACE_IS_FULL,                                                  -- D
DISCRETIONARY_ACCESS_IS_NOT_GRANTED,                                   -- S
ENUMERAL_TYPE_IS_INVALID,                                              -- P
ENUMERAL_TYPE_IS_NOT_IN_ATTRIBUTE_VALUE_TYPE,                          -- P
ENUMERAL_TYPE_IS_NOT_VISIBLE,                                          -- I
```