Standard ECMA-162
4th Edition - December 1997

ECMA

Standardizing Information and Communication Systems

Portable Common Tool Environment (PCTE) - Ada Programming Language Binding

Phone: +41 22 849.60.00 - **Fax:** +41 22 849.60.01 - **URL:** http://www.ecma.ch - **Internet:** helpdesk@ecma.ch

Standard ECMA-162
4th Edition - December 1997

ECMA

Standardizing Information and Communication Systems

Portable Common Tool Environment (PCTE) - Ada Programming Language Binding

Phone: +41 22 849.60.00 - **Fax:** +41 22 849.60.01 - **URL:** http://www.ecma.ch - **Internet:** helpdesk@ecma.ch

IW ECMA-162.DOC 03-02-99 11,42

Brief History

(1) PCTE, Portable Common Tool Environment, is an interface standard. The interface is designed to support program portability by providing machine-independent access to a set of facilities. These facilities, which are described in the PCTE Abstract Specification (Standard ECMA-149), are designed particularly to provide an infrastructure for programs which may be part of environments supporting systems engineering projects. Such programs, which are used as aids to systems development, are often referred to as tools.

(2) PCTE has its origin in the European Strategic Programme for Research and Development in Information Technology (ESPRIT) project 32, called "A Basis for a Portable Common Tool Environment". That project produced a specification for a tool interface, an initial implementation, and some tools on that implementation. The interface specifications were produced in the C Language. A number of versions of the specifications were produced, culminating in the fourth edition known as "PCTE Version 1.4". That was in two volumes; volume 2 covered the user interface and volume 1 covered everything else. Subsequently, the Commission of the European Communities (CEC) commissioned Ada versions of the two volumes of the PCTE specification. Since 1988, a technical committee of ECMA, TC33, has continued the development of PCTE, in a form suitable for standardization under ECMA rules. This work was undertaken by Task Group TGEP (later renamed TGOO) of ECMA TC33, which was formed in November 1988.

(3) The work on the Ada Language Binding for ECMA PCTE was started in mid 1990. The Ada Language Binding was the second binding of ECMA PCTE to be developed, though the strategy for it was developed in parallel with that for the C Language Binding. The text of this binding reflects the desire for the C and Ada Language Bindings to be as compatible as possible.

(4) Following acceptance of the first edition as an ECMA Standard in December 1991, review by international experts led to the production of second edition taking into account review comments relating to this standard and also maintaining consistency with the second edition of Standard ECMA-149. The second edition was accepted by the General Assembly of June 1993, and was submitted as part 3 of the draft PCTE standard to ISO/IEC JTC1 for fast-track processing to international standardization.

(5) During the fast-track processing, which was successfully completed in September 1994, comments from National Bodies resulted in a number of changes to the draft standard. Some further editorial changes were requested by JTC1 ITTF. All these were incorporated in the published international standard, ISO/IEC 13719-3, with which the third edition of this ECMA standard was aligned.

(6) This fourth edition incorporates the resolutions of all comments received too late for consideration during the fast-track processing, or after, and the contents of Standards ECMA-229 (Extensions for Support of Fine-Grain Objects) and ECMA-257 (Object Orientation Extensions). It is aligned with the second edition of ISO/IEC 13719-3.

Adopted as 4th Edition of Standard ECMA-162 by the General Assembly of December 1997.

Contents

.

1 Scope

(1) This ECMA Standard defines the binding of the Portable Common Tool Environment (PCTE) interfaces, as specified in ECMA-149, to the Ada programming language.

(2) A number of features are not completely defined in ECMA-149, 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 ECMA Standard . These constraints are specified in clause 24.

(3) 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

(1) An implementation of PCTE conforms to this ECMA Standard if it conforms to 2.2 of ECMA-149, 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 ECMA Standard. All other parts of this ECMA Standard are provided as assistance to the reader and are not normative.

(2) The Ada language binding defined in this ECMA Standard conforms to 2.1 of ECMA-149.

3 Normative references

(1) The following standards contain provisions which, through reference in this text, constitute provisions of this ECMA Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this ECMA Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below.

(2) ECMA-149 Portable Common Tool Environment (PCTE) - Abstract Specification (4th edition, December 1997)

(3) ECMA-158 Portable Common Tool Environment (PCTE) - C Programming Language Binding (4th edition, December 1997)

(4) ISO 8601 Data elements and interchange formats - Information interchange - Representation of dates and times (1988)

(5) ISO/IEC 8652 Information technology - Programming languages, their environments and system software interfaces - Ada programming language, (referring to ANSI-MIL-STD 1815A - Reference Manual for the Ada Programming Language) (1995)

(6) ISO/IEC/TR 10182 Information technology - Programming languages, their environments and system software interfaces - Guidelines for language bindings (1993)

(7) ISO/IEC 11404 Information technology - Programming languages, their environments and system software interfaces - Language-independent datatypes (1996)

4 Definitions

(1) All technical terms used in this ECMA Standard, other than a few in widespread use, are defined in the body of this ECMA Standard or in the referenced documents.

5 Formal notations

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

6 Outline of the Standard

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

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

(3) Clauses 9 to 22 define the bindings of datatypes and operations in the corresponding clauses of ECMA-149. The extensions for fine-grain objects are added at the end of clause 11.

(4) Clause 23 defines the binding of object and type references, as specified in ECMA-149 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.

(5) Clause 24 defines the binding of the implementation limit subprograms described in ECMA-149, clause 24.

(6) Clause 25 defines the binding of the error conditions specified in ECMA-149, clause 25, and defines binding-defined error conditions for the Ada binding.

(7) Annex A, which is normative, contains the extensions for object orientation, corresponding to annex G of ECMA-149.

7 Binding strategy

7.1 Ada programming language standard

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

7.2 General principles

(1) The following general principles were applied when generating the binding in this ECMA Standard.

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

(3) Each operation in ECMA-149 should be represented by one subprogram in this ECMA Standard unless there are specific reasons to the contrary.

(4) 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 ECMA-149, ECMA-158, and this ECMA Standard.

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

(6) 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 ECMA-158.

(7) 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'

(8) 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'.

(9) Names should be retained from ECMA-149 as far as possible.

(10) All additional names should be chosen appropriately for their meanings.

(11) 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.

(12) 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 ECMA-149.

(13) All simple parameter types in ECMA-149 that represent attribute value types should be mapped to corresponding Ada types defined by this binding.

(14) All simple parameter types in ECMA-149 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

(1) A type defined in this ECMA Standard for which an object is created dynamically is always limited private, and subprograms are provided to construct, access and discard such objects.

(2) 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.

(3) 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.

(4) 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

(1) Some complex entities to be passed into or retrieved from an operation are defined as sets or sequences of a base type in ECMA-149. Where such sets and sequences are bounded and constrainable by the user they are mapped to Ada language array types.

(2) 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 ECMA-149, or by use of a subprogram of the appropriate nested package.

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

(4) 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

(1) 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

(1) This ECMA Standard 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 ECMA Standard , 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

(1) ECMA-149 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

(1) This clause defines the mapping of the parameter and result datatypes of the operations of ECMA-149 (*PCTE datatypes*) to the parameter and result datatypes of the operations of this ECMA Standard (*Ada datatypes*).

(2) PCTE datatype names are printed in normal characters, the names of parameters to operations of ECMA-149 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.

(3) 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

(1) 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.

(2) The general strategy of this mapping is as follows.

(3) - To select for each PCTE datatype a LI datatype definition which matches the requirements of the PCTE datatype defined in ECMA-149. The LI datatype definition is, where possible, a primitive LI datatype, and otherwise a generated LI datatype.

(4) - To define new datatype generators where needed.

(5) - To map PCTE datatypes with the same properties to the same LI datatype.

8.1.1 Mapping of predefined PCTE datatypes

(1) The mapping of these PCTE datatypes is as defined in ECMA-149, clause 23 and is summarized in table 1.

(2) **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

(1) **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*

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

(3) 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

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

(2) 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.

(3) 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.

(4) 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.

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

(6) 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).

(7) 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

(1) 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.

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

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

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

(5) 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.

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

(7) 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.

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

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

(10) 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.

(11) 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

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

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

(14) 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

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

(2) **subtype** boolean **is** STANDARD.BOOLEAN

(3) **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

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

(2) **type** integer **is**
range MIN_INTEGER_ATTRIBUTE .. MAX_INTEGER_ATTRIBUTE;

(3) **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

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

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

(3) **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

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

(2)
```
type float is digits MAX_DIGITS_FLOAT_ATTRIBUTE
    range MIN_FLOAT_ATTRIBUTE .. MAX_FLOAT_ATTRIBUTE;
```

(3) **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

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

(2)
```
type time is private;
```

(3) The range and accuracy of Pcte.calendar.time is implementation-defined but must respect the conditions for conformance with ECMA-149. 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.

(4) **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

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

(2) **Characterizing operations**

Operation	**Ada Operation**
Equal (x,y)	x = y

8.2.7 LI datatype: pcte-text

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

(2) **subtype** text **is** STANDARD.STRING;

(3) **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

(1) 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:

(2) **package** elements **is**

(3) **type** sequence **is limited private**;

```
(4)     -- 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.

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

(6)     -- 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.

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

(8)     -- 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.

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

(10)    -- 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.

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

(12)    -- elements.append appends the element to the sequence.

(13)    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);

(14)    -- 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.
```

(15)
```
    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);
```

(16)
```
    -- 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.
```

(17)
```
    function length_of (
        list         : elements.sequence;
        status       : Pcte_error.handle := EXCEPTION_ONLY)
        return         Pcte.natural;
```

(18)
```
    -- elements.length_of returns the number of elements in the given sequence.
```

(19)
```
    function index_of (
        list         : elements.sequence;
        item         : element;
        status       : Pcte_error.handle := EXCEPTION_ONLY)
        return         Pcte.integer;
```

(20)
```
    -- 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.
```

(21)
```
    function are_equal (
        first        : elements.sequence;
        second       : elements.sequence;
        status       : Pcte_error.handle := EXCEPTION_ONLY)
        return         Pcte.boolean;
```

(22)
```
    -- 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.
```

(23)
```
    procedure normalize (
        list     : in out   elements.sequence;
        status   : in       Pcte_error.handle := EXCEPTION_ONLY);
```

(24)
```
    -- elements.normalize reorders the elements of the given sequence in an
    -- implementation-defined canonical order, and deletes any duplicate elements.
```

(25)
```
    procedure discard (
        list     : in out   elements.sequence;
        status   : in       Pcte_error.handle := EXCEPTION_ONLY);
```

(26)
```
private
```

(27)
implementation-defined

(28) **end** elements;

(29) 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:

(30) - assignment is not supported;

(31) - all elements of such objects always have defined values.

(32) **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

(1) 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:

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

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

(4) **Characterizing operations**

Operation	**Ada Operation**
IsIn (s,e)	s (e)

Subset (s1,s2)	**for** i **in** xxx **loop**
	  if s1(i) **and not** s2(i) **then**
	  return FALSE;
	  end if ;
	end loop ;
	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;
	for i **in** xxx **loop**
	  j := i;
	  exit when s(j);
	end loop ;
	return j;

8.2.10 LI datatype: record

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

(2)
```
type xxx is record
    component_1:    type_1;
    component_2:    type_2;
    component_3:    type_3;
    ...;
end record;
```

(3) **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

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

8.2.12 LI enumerated datatype: pcte-xxx

(1) 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.

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

(3) **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

(1) Each Ada subprogram corresponds to the abstract operation in ECMA-149 whose name precedes the subprogram declaration, as a comment with the form of a clause heading from ECMA-149 e.g.: -- 11.2.9 VOLUME_MOUNT.

(2) 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 ECMA-149, a cross-reference is given.

(3) 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.

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

(5) - Ada formal parameter types correspond to the PCTE datatypes of the corresponding abstract operation parameters as defined in clause 8.

(6) - 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.

(7) - 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.

(8) - 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.

(9) - 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.

(10) - 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.

(11) - 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).

(12) - 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

(1) **with** Pcte_error, CALENDAR;

(2) **package** Pcte is

(3) **use** Pcte_error

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

(4) **subtype** boolean **is** STANDARD.BOOLEAN;

(5) -- Pcte.boolean corresponds to the PCTE datatype Boolean.

(6) **type** integer **is range** *implementation-defined*;

(7) -- Pcte.integer corresponds to the PCTE datatype Integer.

```
(8)  type natural is range 0 .. Pcte.integer'LAST;
(9)  -- Pcte.natural corresponds to the PCTE datatype Natural.
(10) type float is digits implementation-defined range implementation-defined;
(11) -- Pcte.float corresponds to the PCTE datatype Float.
(12) subtype text is STANDARD.STRING;
(13) -- Pcte.text corresponds to the PCTE datatype Text.
(14) subtype string is STANDARD.STRING;
(15) subtype string_length is STANDARD.NATURAL;
(16) -- Pcte.string corresponds to the PCTE datatype String.
(17) subtype key is Pcte.text;
(18) -- Pcte.key corresponds to the PCTE datatype Key.
(19) subtype actual_key is Pcte.text;
(20) -- Pcte.actual_key corresponds to the PCTE datatype Actual_key.
(21) subtype link_name is Pcte.text;
(22) -- Pcte.link_name corresponds to the PCTE datatype Link_name.
(23) subtype name is Pcte.text;
(24) -- Pcte.name corresponds to the PCTE datatype Name.
(25) subtype type_name is Pcte.text;
(26) -- Pcte.type_name corresponds to the PCTE datatype Type_name.
(27) subtype type_name_in_sds is Pcte.text (1..MAX_NAME_SIZE);
(28) -- Pcte.type_name corresponds to the PCTE datatype Type_name_in_sds.
(29) subtype positive is STANDARD.POSITIVE;
(30) -- Pcte.positive is used to define a character position within Pcte.string or Pcte.text.
(31) package calendar is
(32)     type time is private;
(33)     -- 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.
(34)     DEFAULT_TIME: constant time;
(35)     Pcte.calendar.DEFAULT_TIME is the default value of time attributes.
(36)     subtype year_number is STANDARD.CALENDAR.YEAR_NUMBER
                     range 1980 .. 2044;
(37)     subtype month_number is STANDARD.CALENDAR.MONTH_NUMBER;
(38)     subtype day_number is STANDARD.CALENDAR.DAY_NUMBER;
```

```
(39)  subtype day_duration is STANDARD.CALENDAR.DAY_DURATION;
(40)  function clock return Pcte.calendar.time;
(41)  function year (
          date      : Pcte.calendar.time)
          return      Pcte.calendar.year_number;
(42)  function month (
          date      : Pcte.calendar.time)
          return      Pcte.calendar.month_number;
(43)  function day  (
          date      : Pcte.calendar.time)
          return      Pcte.calendar.day_number;
(44)  function seconds (
          date      : Pcte.calendar.time)
          return      Pcte.calendar.day_duration;
(45)  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);
(46)  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;
(47)  function "+" (
          left      : Pcte.calendar.time;
          right     : STANDARD.DURATION)
          return      Pcte.calendar.time;
(48)  function "+" (
          left      : STANDARD.DURATION;
          right     : Pcte.calendar.time)
          return      Pcte.calendar.time;
(49)  function "-" (
          left      : Pcte.calendar.time;
          right     : STANDARD.DURATION)
          return      Pcte.calendar.time;
(50)  function "-" (
          left      : Pcte.calendar.time;
          right     : Pcte.calendar.time)
          return      STANDARD.DURATION;
```

```
(51)        function "<" (
                left        : Pcte.calendar.time;
                right       : Pcte.calendar.time)
                return      Pcte.boolean;
(52)        function "<=" (
                left        : Pcte.calendar.time;
                right       : Pcte.calendar.time)
                return      Pcte.boolean;
(53)        function ">" (
                left        : Pcte.calendar.time;
                right       : Pcte.calendar.time)
                return      Pcte.boolean;
(54)        function ">=" (
                left        : Pcte.calendar.time;
                right       : Pcte.calendar.time)
                return      Pcte.boolean;
(55)        function extend (
                date        : Pcte.calendar.time)
                return      STANDARD.CALENDAR.TIME;
(56)        -- Pcte.calendar.extend converts the value of the parameter date to the type
            -- CALENDAR.TIME.
(57)        function round (
                date        : STANDARD.CALENDAR.TIME)
                return      Pcte.calendar.time;
(58)        -- Pcte.calendar.round converts the value of the parameter date to the type
            -- Pcte.calendar.time.
(59)        TIME_ERROR : exception renames STANDARD.CALENDAR.TIME_ERROR;
(60)    private
(61)        implementation-defined
(62)    end calendar;
```

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

```
(63)    package reference is
(64)        type  object_ref     is limited private;
(65)        type  link_ref       is limited private;
(66)        type  type_ref       is limited private;
(67)        subtype attribute_ref is type_ref;
(68)        CURRENT_PROCESS          : constant Pcte.reference.object_ref;
(69)        LOCAL_WORKSTATION        : constant Pcte.reference.object_ref;
(70)        LOCAL_EXECUTION_SITE     : constant Pcte.reference.object_ref;
(71)        subtype pathname is Pcte.string;
```

```
(72)    -- Pcte.reference.pathname corresponds to the PCTE datatype Pathname.
(73)    subtype relative_pathname is Pcte.string;
(74)    -- Pcte.reference.relative_pathname corresponds to the PCTE datatype
        -- Relative_pathname.
(75)    type evaluation_point is (NOW, FIRST_USE, EVERY_USE);
(76)    -- Pcte.reference.evaluation_point corresponds to the PCTE datatype
        -- Evaluation_point.
(77)    type evaluation_status is (INTERNAL, EXTERNAL, UNDEFINED);
(78)    -- 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.
(79)    type reference_equality is (EQUAL_REFS, UNEQUAL_REFS, EXTERNAL_REFS);
(80)    -- Pcte.reference.reference_equality corresponds to the PCTE datatype
        -- Reference_equality.
(81)    type key_kind is (NATURAL_KEY, STRING_KEY);
(82)    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
(83)    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
(84)    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
(85)    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

(86)    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

(87)    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

(88)    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

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

        -- 23.2.8 OBJECT_REFERENCES_ARE_EQUAL

(90)    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

(91)    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

(92)    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

(93)  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

(94)  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

(95)  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

(96)  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

(97)  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

(98)  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);

(99)  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);
```

```
(100)   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

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

        -- 23.3.10 LINK_REFERENCES_ARE_EQUAL

(102)   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

(103)   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

(104)   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

(105)   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

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

```
(107)       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

(108)       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

(109)       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

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

            -- 23.4.8 TYPE_REFERENCES_ARE_EQUAL

(111)       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;

(112)   private

(113)       implementation-defined

(114)   end reference;


(115)   subtype object_reference is Pcte.reference.object_ref;

(116)   subtype type_reference is Pcte.reference.type_ref;

(117)   subtype link_reference is Pcte.reference.link_ref;

(118)   subtype attribute_reference is Pcte.reference.attribute_ref;

(119)   -- These types correspond to the PCTE datatypes X_reference.

    ---------------------------------  PCTE types related to attributes  ---------------------------------

(120)   type enumeral_position is range 0 .. Pcte.natural'LAST;

(121)   type value_type is (INTEGER_TYPE, NATURAL_TYPE, BOOLEAN_TYPE,
            TIME_TYPE, FLOAT_TYPE, STRING_TYPE, ENUMERAL_TYPE);
```

```
(122)   -- 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.

(123)   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;

(124)   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;

(125)   --      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  ----------------------------------

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

(127)   -- Pcte.category corresponds to the PCTE datatype Category.

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

(129)   -- Pcte.categories corresponds to the PCTE datatype Categories.

(130)   type object_scope is (ATOMIC, COMPOSITE);

(131)   -- Pcte.object_scope corresponds to the PCTE datatype Object_scope.

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

(132)   subtype exact_identifier is Pcte.text;
```

```
(133)   -- Pcte.exact_identifier corresponds to the PCTE datatype Exact_identifier.

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

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


(135)   package object_references is

(136)       type sequence is limited private;

(137)       -- Pcte.object_references.sequence corresponds to the PCTE datatype Object_references.

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

(139)       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);

(140)       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);

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

(142)       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);

(143)       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);

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

```
(145)       function index_of (
                list        : Pcte.object_references.sequence;
                item        : Pcte.object_reference;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.integer;
(146)       function are_equal (
                first       : Pcte.object_references.sequence;
                second      : Pcte.object_references.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.boolean;
(147)       procedure normalize (
                list        : in out    Pcte.object_references.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(148)       procedure discard (
                list        : in out    Pcte.object_references.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(149)   private
(150)       implementation-defined
(151)   end object_references;

(152)   package link_references is
(153)       type sequence is limited private;
(154)       -- Pcte.link_references.sequence corresponds to the PCTE datatype Link_references.
(155)       function get (
                list        : Pcte.link_references.sequence;
                index       : Pcte.natural := Pcte.natural'FIRST;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.link_reference;
(156)       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);
(157)       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);
(158)       procedure append (
                list        : in out    Pcte.link_references.sequence;
                item        : in        Pcte.link_reference;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
(159)       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);
(160)       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);
(161)       function length_of (
                list        : Pcte.link_references.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.natural;
(162)       function index_of (
                list        : Pcte.link_references.sequence;
                item        : Pcte.link_reference;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.integer;
(163)       function are_equal (
                first       : Pcte.link_references.sequence;
                second      : Pcte.link_references.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.boolean;
(164)       procedure normalize (
                list        : in out    Pcte.link_references.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(165)       procedure discard (
                list        : in out    Pcte.link_references.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(166)   private
(167)       implementation-defined
(168)   end link_references;

(169)   package type_references is
(170)       type sequence is limited private;
(171)       -- Pcte.type_references.sequence corresponds to the PCTE datatype Type_references.
```

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

(173)   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);

(174)   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);

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

(176)   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);

(177)   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);

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

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

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

```
(181)       procedure normalize (
                list        : in out    Pcte.type_references.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(182)       procedure discard (
                list        : in out    Pcte.type_references.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(183)   private
(184)       implementation-defined
(185)   end type_references;

(186)   package type_names_in_sds is
(187)       type sequence is limited private;
(188)       -- Pcte.type_names_in_sds.sequence corresponds to the PCTE datatype
            -- Type_names_in_sds.
(189)       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;
(190)       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);
(191)       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);
(192)       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);
(193)       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);
```

```
(194)       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);
(195)       function length_of (
                list          : Pcte.type_names_in_sds.sequence;
                status        : Pcte_error.handle := EXCEPTION_ONLY)
                return          Pcte.natural;
(196)       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;
(197)       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;
(198)       procedure normalize (
                list          : in out   Pcte.type_names_in_sds.sequence;
                status        : in       Pcte_error.handle := EXCEPTION_ONLY);
(199)       procedure discard (
                list          : in out   Pcte.type_names_in_sds.sequence;
                status        : in       Pcte_error.handle := EXCEPTION_ONLY);
(200)   private
(201)       implementation-defined
(202)   end type_names_in_sds;


(203)   package attribute_references is
(204)       type sequence is limited private;
(205)       -- Pcte.attribute_references.sequence corresponds to the PCTE datatype
            -- Attribute_references.
(206)       function get (
                list          : Pcte.attribute_references.sequence;
                index         : Pcte.natural := Pcte.natural'FIRST;
                status        : Pcte_error.handle := EXCEPTION_ONLY)
                return          Pcte.attribute_reference;
```

```
(207)   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);

(208)   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);

(209)   procedure append (
            list      : in out   Pcte.attribute_references.sequence;
            item      : in       Pcte.attribute_reference;
            status    : in       Pcte_error.handle := EXCEPTION_ONLY);

(210)   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);

(211)   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);

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

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

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

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

```
(216)   procedure discard (
            list        : in out    Pcte.attribute_references.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(217)   private
(218)       implementation-defined
(219)   end attribute_references;

(220)   package attribute_assignments is
(221)       type sequence is limited private;
(222)       -- Pcte.attribute_assignments.sequence corresponds to the PCTE datatype
            -- Attribute_assignments.
(223)       function get (
            list        : Pcte.attribute_assignments.sequence;
            index       : Pcte.natural := Pcte.natural'FIRST;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.attribute_assignment;
(224)       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);
(225)       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);
(226)       procedure append (
            list        : in out    Pcte.attribute_assignments.sequence;
            item        : in        Pcte.attribute_assignment;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(227)       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);
(228)       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);
```

```
(229)       function length_of (
                list        : Pcte.attribute_assignments.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.natural;
(230)       function index_of (
                list        : Pcte.attribute_assignments.sequence;
                item        : Pcte.attribute_assignment;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.integer;
(231)       function are_equal (
                first       : Pcte.attribute_assignments.sequence;
                second      : Pcte.attribute_assignments.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.boolean;
(232)       procedure normalize (
                list        : in out    Pcte.attribute_assignments.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(233)       procedure discard (
                list        : in out    Pcte.attribute_assignments.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(234)   private
(235)       implementation-defined
(236)   end attribute_assignments;
(237) end Pcte;
```

9 Object managment

9.1 Object management datatypes

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

9.2 Link operations

```
(1)   with Pcte, Pcte_error, Pcte_archive, Pcte_volume;
(2)   package Pcte_link is
(3)       use Pcte, Pcte.calendar, Pcte_error;

          -- 9.2.1 LINK_CREATE

(4)       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

(5)     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

(6)     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

(7)     -- 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.

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

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

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

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

(12)
```
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;
```

(13)
```
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;
```

(14)
```
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
```

(15)
```
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
```

(16)
```
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
```

(17)
```
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
```

(18)
```
-- 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.
```

(19)
```
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);
```

(20)
```
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

(21)
```
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

(22)
```
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

(23)
```
-- 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.
```

(24)
```
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);
```

(25)
```
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);
```

```
(26)    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);
(27)    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);
(28)    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);
(29)    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);
(30)    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

(31)    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

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

(33)  end Pcte_link;
```

9.3 Object operations

```
(1)  with Pcte, Pcte_error, Pcte_discretionary, Pcte_volume;
(2)  package Pcte_object is
(3)      use Pcte, Pcte.calendar, Pcte_error;
(4)      type type_ancestry is (EQUAL_TYPE, ANCESTOR_TYPE, DESCENDANT_TYPE,
             UNRELATED_TYPE);
(5)      -- Pcte_object.type_ancestry corresponds to the PCTE datatype Type_ancestry.
(6)      type link_scope is (INTERNAL_LINKS, EXTERNAL_LINKS, ALL_LINKS);
(7)      -- Pcte_object.link_scope corresponds to the PCTE datatype Link_scope.
(8)      type link_set_descriptor is record
             origin  : Pcte.object_reference;
             links   : Pcte.link_references.sequence;
         end record;
(9)      -- Pcte_object.link_set_descriptor corresponds to the PCTE datatype Link_set_descriptor.

(10)     -- The semantics of the operations of this package are defined in 8.2.8.
(11)     package link_set_descriptors is
(12)         type sequence is limited private;
(13)         -- Pcte_object.link_set_descriptor.sequence corresponds to the PCTE datatype
             -- Link_set_descriptors.
(14)         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;
(15)         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);
(16)         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);
(17)         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);
```

```
(18)        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);
(19)        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);
(20)        function length_of (
                list        : Pcte_object.link_set_descriptors.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.natural;
(21)        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;
(22)        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;
(23)        procedure normalize (
                list        : in out    Pcte_object.link_set_descriptors.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(24)        procedure discard (
                list        : in out    Pcte_object.link_set_descriptors.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(25)    private
(26)        implementation-defined
(27)    end link_set_descriptors;

(28)    type time_kind is (CURRENT_SYSTEM_TIME, SPECIFIED_TIME);
(29)    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

(30)  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

(31)  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

(32)  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);

(33)  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

(34)  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);
```

(35)
```
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

(36)
```
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

(37)
```
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

(38)
```
-- 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.
```

(39)
```
function get_attribute (
    object     : Pcte.object_reference;
    attribute  : Pcte.attribute_reference;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.boolean;
```

(40)
```
function get_attribute (
    object     : Pcte.object_reference;
    attribute  : Pcte.attribute_reference;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.natural;
```

(41)
```
function get_attribute (
    object     : Pcte.object_reference;
    attribute  : Pcte.attribute_reference;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.integer;
```

(42)
```
function get_attribute (
    object     : Pcte.object_reference;
    attribute  : Pcte.attribute_reference;
    status     : Pcte_error.handle := EXCEPTION_ONLY)
    return       Pcte.float;
```

(43)
```
function get_attribute (
    object      : Pcte.object_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.calendar.time;
```

(44)
```
function get_attribute (
    object      : Pcte.object_reference;
    attribute   : Pcte.attribute_reference;
    status      : Pcte_error.handle := EXCEPTION_ONLY)
    return      Pcte.string;
```

(45)
```
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

(46)
```
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);
```

(47)
```
-- 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

(48)
```
-- 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.
```

(49)
```
procedure get_several_attributes (
    object : in         Pcte.object_reference;
    values : in out     Pcte.attribute_assignments.sequence;
    status : in         Pcte_error.handle := EXCEPTION_ONLY);
```

(50)
```
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

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

-- 9.3.11 OBJECT_IS_COMPONENT

(52)  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

(53)  -- 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.

(54)  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);

(55)  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);

(56)  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

(57)  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

(58)  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

(59)  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

(60)  -- 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.

(61)  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);

(62)  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);

(63)  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);

(64)  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);

(65)  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);
```

(66)
```
procedure set_attribute (
    object     : in    Pcte.object_reference;
    attribute  : in    Pcte.attribute_reference;
    value      : in    Pcte.string := "";
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);
```

(67)
```
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

(68)
```
-- 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.
```

(69)
```
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);
```

(70)
```
procedure set_type_preference (
    object     : in    Pcte.object_reference;
    pcte_type  : in    Pcte.type_reference;
    status     : in    Pcte_error.handle := EXCEPTION_ONLY);
```

(71)
```
procedure set_key_preference (
    object : in    Pcte.object_reference;
    key    : in    Pcte.string;
    status : in    Pcte_error.handle := EXCEPTION_ONLY);
```

(72)
```
procedure unset_preference (
    object : in    Pcte.object_reference;
    status : in    Pcte_error.handle := EXCEPTION_ONLY);
```

-- 9.3.18 OBJECT_SET_SEVERAL_ATTRIBUTES

(73)
```
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
(74)    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
(75)    --  See 11.2.

(76)  end   Pcte_object;
```

9.4 Version operations

```
(1)   with Pcte, Pcte_error, Pcte_discretionary;

(2)   package Pcte_version is

(3)       use Pcte, Pcte_error;

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

(5)       --  Pcte_version.version_relation corresponds to the PCTE datatype Version_relation.

          -- 9.4.1 VERSION_ADD_PREDECESSOR

(6)       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

(7)       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

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

```
-- 9.4.4 VERSION_REMOVE_PREDECESSOR

(9)  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

(10) 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);

(11) 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

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

(13) 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);

(14) 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);
```

```
(15)    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);
(16)    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

(17)    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

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

(19)  end  Pcte_version;
```

10 Schema management

```
(1)   with Pcte, Pcte_error;

(2)   package Pcte_sds is

(3)      use Pcte, Pcte_error, Pcte.calendar, Pcte.reference;
```

10.1 Schema management datatypes

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

(2)      --  Pcte_sds.duplication corresponds to the PCTE datatype Duplication.

(3)      type exclusiveness is (SHARABLE, EXCLUSIVE);

(4)      --  Pcte_sds.exclusiveness corresponds to the PCTE datatype Exclusiveness.

(5)      type stability is (ATOMIC_STABLE, COMPOSITE_STABLE, NON_STABLE);

(6)      --  Pcte_sds.stability corresponds to the PCTE datatype Stability.

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

(8) -- 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.

(9) **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;

(10) -- 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.

(11) **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;

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

(13) **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;

(14) -- Pcte_sds.object_type_properties corresponds to the PCTE datatypes of the results of
-- SDS_GET_OBJECT_TYPE_PROPERTIES.

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

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

(17) -- Pcte_sds.definition_mode_values corresponds to the PCTE datatype
-- Definition_mode_values.

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

(19) -- Pcte_sds.attribute_scan_kind corresponds to the PCTE datatype Attribute_scan_kind.

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

(21) -- Pcte_sds.link_scan_kind corresponds to the PCTE datatype Link_scan_kind.

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

```
(23)    -- Pcte_sds.object_scan_kind corresponds to the PCTE datatype Object_scan_kind.
(24)    type type_kind is (OBJECT_TYPE, LINK_TYPE, ATTRIBUTE_TYPE,
            ENUMERAL_TYPE);
(25)    -- Pcte_sds.type_kind corresponds to the PCTE datatype Type_kind.
```

10.2 Update operations

```
        -- 10.2.1 SDS_ADD_DESTINATION
(1)     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
(2)     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
(3)     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
(4)     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

(5)  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

(6)  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

(7)  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

(8)  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

(9)  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

(10) 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

(11) 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

(12) 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

(13)  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

(14)  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

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

-- 10.2.16 SDS_IMPORT_ATTRIBUTE_TYPE

(16)  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

(17)  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

(18)    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

(19)    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

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

        -- 10.2.21 SDS_REMOVE

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

        -- 10.2.22 SDS_REMOVE_DESTINATION

(22)    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

(23)    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

(24)    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

(25)    -- 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.

(26)    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);

(27)    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);

(28)    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

(29)    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

(30)    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

(31)    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

```
(1)   -- 10.3.1 SDS_GET_ATTRIBUTE_TYPE_PROPERTIES

(2)   -- 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.

(3)   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

(4)   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

(5)   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

(6)   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

(7)   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

(8)  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

(9)  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

(10) 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

(11) 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

(12) 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

(13) 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

(14)  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

(15)  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);

(16)  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

(1)   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

(2)   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

(3)   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

(4)   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

(5)  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

(6)  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

(7)  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

(8)  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

(9)  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

(10) 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

(11) 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
(12)    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
(13)    procedure scan_types (
            kind   : in      Pcte_sds.type_kind;
            types  : in out  Pcte.type_references.sequence;
            status : in      Pcte_error.handle := EXCEPTION_ONLY);
(14)    procedure scan_types (
            types  : in out  Pcte.type_references.sequence;
            status : in      Pcte_error.handle := EXCEPTION_ONLY);
(15) end Pcte_sds;
```

11 Volumes, devices, and archives

11.1 Volume, device, and archive datatypes

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

11.2 Volume, device, and archive operations

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

         -- 11.2.1 ARCHIVE_CREATE
(8)      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
(9)     procedure remove (
            archive    : in    Pcte.object_reference;
            status     : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 11.2.3 ARCHIVE_RESTORE
(10)    -- 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).

(11)    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);

(12)    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

(13)    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);

(14) end Pcte_archive;


(15) with Pcte, Pcte_error, Pcte_contents, Pcte_discretionary;

(16) package Pcte_device is

(17)    use Pcte, Pcte_error;

(18)    type device_identifier is new Pcte.natural;

(19)    -- Pcte_device.device_identifier corresponds to the PCTE datatype Device_identifier.
```

```
        -- 11.2.5 DEVICE_CREATE
(20)    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
(21)    procedure remove (
            device    : in   Pcte.object_reference;
            status    : in   Pcte_error.handle := EXCEPTION_ONLY);

        -- 11.2.7 LINK_GET_DESTINATION_ARCHIVE
(22)    --  See 9.2.

        -- 12.2.13 DEVICE_GET_CONTROL
(23)    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
(24)    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);
(25)    end Pcte_device;


(26)    with Pcte, Pcte_error, Pcte_discretionary;
(27)    package Pcte_volume is
(28)        use Pcte, Pcte_error;
(29)        type volume_accessibility is (ACCESSIBLE, INACCESSIBLE, UNKNOWN);
(30)        -- Pcte_volume.volume_accessibility corresponds to the PCTE datatype
            -- Volume_accessibility.
(31)        type volume_identifier is new Pcte.natural;
```

```
(32)    -- Pcte_volume.volume_identifier corresponds to the PCTE datatype Volume_identifier.
(33)    type volume_info is record
            identity    : Pcte_volume.volume_identifier;
            mounted     : Pcte_volume.volume_accessibility;
        end record;
(34)    -- Pcte_volume.volume_info corresponds to the PCTE datatype Volume_info.
(35)    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;
(36)    -- Pcte_volume.volume_status corresponds to the PCTE datatype Volume_status.

(37)    -- The semantics of the operations of this package are defined in 8.2.8.
(38)    package volume_infos is
(39)        type sequence is limited private;
(40)        -- Pcte_volume.volume_infos.sequence corresponds to the PCTE datatype
            -- Volume_infos.
(41)        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;
(42)        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);
(43)        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);
(44)        procedure append (
                list        : in out    Pcte_volume.volume_infos.sequence;
                item        : in        Pcte_volume.volume_info;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
(45)    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);
(46)    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);
(47)    function length_of (
            list        : Pcte_volume.volume_infos.sequence;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.natural;
(48)    function index_of (
            list        : Pcte_volume.volume_infos.sequence;
            item        : Pcte_volume.volume_info;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.integer;
(49)    function are_equal (
            first       : Pcte_volume.volume_infos.sequence;
            second      : Pcte_volume.volume_infos.sequence;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.boolean;
(50)    procedure normalize (
            list    : in out    Pcte_volume.volume_infos.sequence;
            status  : in        Pcte_error.handle := EXCEPTION_ONLY);
(51)    procedure discard (
            list    : in out    Pcte_volume.volume_infos.sequence;
            status  : in        Pcte_error.handle := EXCEPTION_ONLY);
(52)    private
(53)        implementation-defined
(54)    end volume_infos;

        -- 11.2.8 VOLUME_CREATE

(55)    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
(56)    procedure delete (
            volume    : in    Pcte.object_reference;
            status    : in    Pcte_error.handle := EXCEPTION_ONLY);

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

        -- 11.2.11 VOLUME_MOUNT
(58)    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
(59)    procedure unmount (
            volume    : in    Pcte.object_reference;
            status    : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 9.3.20 VOLUME_LIST_OBJECTS
(60)    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);
(61)  end Pcte_volume;


(62)  with Pcte, Pcte_error, Pcte_discretionary;
(63)  package Pcte_cluster is
(64)    use Pcte, Pcte_error;
        -- 11.3.1 CLUSTER_CREATE
(65)    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
(66)    procedure delete (
            cluster     : in    Pcte.object_reference;
            status      : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 11.3.3 CLUSTER_LIST_OBJECTS
(67)    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);
(68) end Pcte_cluster;
```

12 Files, pipes, and devices

```
(1)  with Pcte, Pcte_error;
(2)  package Pcte_contents is
(3)      use Pcte, Pcte_error;
```

12.1 File, pipe, and device datatypes

```
(1)      type contents_access_mode is (READ_WRITE, READ_ONLY, WRITE_ONLY,
             APPEND_ONLY);
(2)      -- Pcte_contents.contents_access_mode corresponds to the PCTE datatype
         -- Contents_access_mode.
(3)      type seek_position is (FROM_BEGINNING, FROM_CURRENT, FROM_END);
(4)      -- Pcte_contents.seek_position corresponds to the PCTE datatype Seek_position.
(5)      type pcte_set_position is (AT_BEGINNING, AT_POSITION, AT_END);
(6)      -- Pcte_contents.pcte_set_position corresponds to the PCTE datatype Set_position.
(7)      type positioning_style is (SEQUENTIAL, DIRECT, SEEK);
(8)      -- Pcte_contents.positioning_style corresponds to the PCTE datatype Positioning_style.
(9)      type position_handle is limited private;
(10)     -- Pcte_contents.position_handle corresponds to the PCTE datatype Position_handle.
(11)     type contents_handle is limited private;
(12)     -- Pcte_contents.contents_handle corresponds to the PCTE datatype Contents_handle.
```

12.2 File, pipe, and device operations

```
(1)      -- 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

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

-- 12.2.2 CONTENTS_GET_HANDLE_FROM_KEY

(3)  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

(4)  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

(5)  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

(6)  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);

(7)  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

(8)  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
(9)     function read (
            contents   : Pcte_contents.contents_handle;
            size       : Pcte.natural;
            status     : Pcte_error.handle := EXCEPTION_ONLY)
            return       Pcte.string;

        -- 12.2.8 CONTENTS_SEEK
(10)    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
(11)    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
(12)    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
(13)    procedure truncate (
            contents   : in   Pcte_contents.contents_handle;
            status     : in   Pcte_error.handle := EXCEPTION_ONLY);

        -- 12.2.12 CONTENTS_WRITE
(14)    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
(15)    --  See 11.2.

        -- 12.2.14 DEVICE_SET_CONTROL
(16)    --  See 11.2.
```

```
        -- 18.3.1 CONTENTS_COPY_FROM_FOREIGN_SYSTEM
(17)    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
(18)    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);
(19)  private
(20)    implementation-defined
(21)  end Pcte_contents;
```

13 Process execution

```
(1)   with Pcte, Pcte_error, Pcte_mandatory, Pcte_discretionary, SYSTEM;
(2)   package Pcte_process is
(3)     use Pcte, Pcte_error, Pcte.reference;
```

13.1 Process execution datatypes

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

```
(13)  -- The semantics of the operations of this package are defined in 8.2.8.
(14)  package profile_buffer is
(15)      type sequence is limited private;
(16)      -- Pcte_process.profile_buffer.sequence corresponds to the PCTE datatype Buffer.
(17)      function get (
              list      : Pcte_process.profile_buffer.sequence;
              index     : Pcte.natural := Pcte.natural'FIRST;
              status    : Pcte_error.handle := EXCEPTION_ONLY)
              return      Pcte.natural;
(18)      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);
(19)      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);
(20)      procedure append (
              list      : in out  Pcte_process.profile_buffer.sequence;
              item      : in      Pcte.natural;
              status    : in      Pcte_error.handle := EXCEPTION_ONLY);
(21)      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);
(22)      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);
(23)      function length_of (
              list      : Pcte_process.profile_buffer.sequence;
              status    : Pcte_error.handle := EXCEPTION_ONLY)
              return      Pcte.natural;
(24)      function index_of (
              list      : Pcte_process.profile_buffer.sequence;
              item      : Pcte.natural;
              status    : Pcte_error.handle := EXCEPTION_ONLY)
              return      Pcte.integer;
```

```
(25)        function are_equal (
                first       : Pcte_process.profile_buffer.sequence;
                second      : Pcte_process.profile_buffer.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.boolean;
(26)        procedure normalize (
                list        : in out    Pcte_process.profile_buffer.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(27)        procedure discard (
                list        : in out    Pcte_process.profile_buffer.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(28)    private
(29)        implementation-defined
(30)    end profile_buffer;

(31)    -- The semantics of the operations of this package are defined in 8.2.8.
(32)    package names is
(33)        type sequence is limited private;
(34)        -- Pcte_process.names.sequence corresponds to the PCTE datatype Name_sequence.
(35)        function get (
                list        : Pcte_process.names.sequence;
                index       : Pcte.natural := Pcte.natural'FIRST;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.name;
(36)        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);
(37)        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);
(38)        procedure append (
                list        : in out    Pcte_process.names.sequence;
                item        : in        Pcte.name;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
(39)        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);
(40)        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);
(41)        function length_of (
                list        : Pcte_process.names.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.natural;
(42)        function index_of (
                list        : Pcte_process.names.sequence;
                item        : Pcte.name;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.integer;
(43)        function are_equal (
                first       : Pcte_process.names.sequence;
                second      : Pcte_process.names.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.boolean;
(44)        procedure normalize (
                list        : in out    Pcte_process.names.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(45)        procedure discard (
                list        : in out    Pcte_process.names.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(46)    private
(47)        implementation-defined
(48)    end names;
```

13.2 Process execution

```
        -- 13.2.1 PROCESS_CREATE
(1)     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
(2)     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
(3)     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
(4)     procedure interrupt_operation (
            process    : in    Pcte.object_reference;
            status     : in    Pcte_error.handle := EXCEPTION_ONLY);

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

        -- 13.2.6 PROCESS_SET_ALARM
(6)     procedure set_alarm (
            duration   : in    Pcte.natural;
            status     : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
        -- 13.2.7 PROCESS_SET_FILE_SIZE_LIMIT
(7)     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
(8)     procedure set_operation_time_out (
            duration    : in    Pcte.natural;
            status      : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 13.2.9 PROCESS_SET_PRIORITY
(9)     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
(10)    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
(11)    procedure set_termination_status (
            termination_status  : in    Pcte.integer;
            status              : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 13.2.12 PROCESS_SET_WORKING_SCHEMA
(12)    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
(13)    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);
```

```
(14)    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

(15)    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

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

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

        -- 13.2.16 PROCESS_UNSET_REFERENCED_OBJECT

(18)    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

(19)    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

(20)    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

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

        -- 22.3.2 PROCESS_UNSET_CONSUMER_IDENTITY

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

13.3 Security operations

```
        -- 13.3.1 PROCESS_ADOPT_USER_GROUP

(1)     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

(2)     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

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

        -- 13.3.4 PROCESS_SET_ADOPTABLE_FOR_CHILD

(4)     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

(5)     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

(6)     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

(7)     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

(8)  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

(9)  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

(10) 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

(11) 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

(1)  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

(2)  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
(1)     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
(2)     procedure continue (
            process  : in   Pcte.object_reference;
            status   : in   Pcte_error.handle := EXCEPTION_ONLY);

        -- 13.5.3 PROCESS_PEEK
(3)     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
(4)     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
(5)     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
(6)     procedure wait_for_breakpoint (
            process     : in    Pcte.object_reference;
            breakpoint  : out   Pcte_process.address;
            status      : in    Pcte_error.handle := EXCEPTION_ONLY);
(7)     private
(8)         implementation-defined
(9)     end Pcte_process;
```

14 Message queues

```
(1)  with Pcte, Pcte_error;
(2)  package Pcte_message is
(3)      use Pcte_error;
```

14.1 Message queue datatypes

```
(1)      type standard_message_type is (INTERRUPT_MSG, QUIT_MSG, FINISH_MSG,
             SUSPEND_MSG, END_MSG, ABORT_MSG, DEADLOCK_MSG, WAKE_MSG);
(2)      -- Pcte_message.standard_message_type corresponds to the PCTE datatype
         -- Standard_message_type.
(3)      type notification_message_type is (MODIFICATION_MSG, CHANGE_MSG,
                 DELETE_MSG, MOVE_MSG, NOT_ACCESSIBLE_MSG, LOST_MSG);
(4)      -- Pcte_message.notification_message_type corresponds to the PCTE datatype
         -- Notification_message_type.
(5)      type implementation_defined_message_type is new Pcte.natural;
(6)      -- Pcte_message.implementation_defined_message_type corresponds to the PCTE
         -- datatype Implementation_defined_message_type.
(7)      type undefined_message_type is new Pcte.natural;
(8)      -- Pcte_message.undefined_message_type corresponds to the PCTE datatype
         -- Undefined_message_type.
(9)      type message_type_kind is (STANDARD_MESSAGE,
             NOTIFICATION_MESSAGE, IMPLEMENTATION_DEFINED_MESSAGE,
             UNDEFINED_MESSAGE);
(10)     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;
```

```
(11)    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;

(12)    -- Pcte_message.received_message corresponds to the PCTE datatype Received_message.

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

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

(15)    subtype send_mode is Pcte_message.receive_mode range NO_WAIT .. WAIT;

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


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

(18)    package message_types is

(19)        type sequence is limited private;

(20)        -- Pcte_message.message_types.sequence corresponds to the PCTE
            -- datatype Message_types.

(21)        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;

(22)        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);

(23)        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);

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

```
(25)    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);
(26)    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);
(27)    function length_of (
            list      : Pcte_message.message_types.sequence;
            status    : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.natural;
(28)    function index_of (
            list      : Pcte_message.message_types.sequence;
            item      : Pcte_message.message_type;
            status    : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.integer;
(29)    function are_equal (
            first     : Pcte_message.message_types.sequence;
            second    : Pcte_message.message_types.sequence;
            status    : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.boolean;
(30)    procedure normalize (
            list      : in out   Pcte_message.message_types.sequence;
            status    : in       Pcte_error.handle := EXCEPTION_ONLY);
(31)    procedure discard (
            list      : in out   Pcte_message.message_types.sequence;
            status    : in       Pcte_error.handle := EXCEPTION_ONLY);
(32)  private
(33)    implementation-defined
(34)  end message_types;
```

14.2 Message queue operations

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

```
        -- 14.2.2 MESSAGE_PEEK

(2)     -- 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.

(3)     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;

(4)     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

(5)     -- 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

(6)     -- 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

(7)     -- 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.

(8)     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

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

(10)    end Pcte_message;


(11)    with Pcte, Pcte_error, Pcte_message;
```

```
(12)  package Pcte_queue is
(13)      use Pcte_error, Pcte_message;

          -- 14.2.7 QUEUE_EMPTY

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

          -- 14.2.8 QUEUE_HANDLER_DISABLE

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

          -- 14.2.9 QUEUE_HANDLER_ENABLE

(16)      -- 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.

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

(18)      package handlers is

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

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

(21)      end handlers;

          -- 14.2.10 QUEUE_RESERVE

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

          -- 14.2.11 QUEUE_RESTORE

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

```
        -- 14.2.12 QUEUE_SAVE
(24)    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
(25)    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
(26)    procedure unreserve (
            queue  : in    Pcte.object_reference;
            status : in    Pcte_error.handle := EXCEPTION_ONLY);
(27) end Pcte_queue;
```

15 Notification

```
(1)  with Pcte, Pcte_error, Pcte_message;
(2)  package Pcte_notify is
(3)      use Pcte_error;
```

15.1 Notification datatypes

```
(1)      type access_event is (MODIFICATION_EVENT, CHANGE_EVENT, DELETE_EVENT,
             MOVE_EVENT);
(2)      -- Pcte_notify.access_event corresponds to the PCTE datatype Access_event.
(3)      type access_events is array (Pcte_notify.access_event) of Pcte.boolean;
(4)      -- Pcte_notify.access_events corresponds to the PCTE datatype Access_events.
```

15.2 Notification operations

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

```
        -- 15.2.2 NOTIFY_CREATE
(2)     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
(3)     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
(4)     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);
(5) end Pcte_notify;
```

16 Concurrency and integrity control

```
(1) with Pcte, Pcte_error;
(2) package Pcte_activity is
(3)     use Pcte_error;
```

16.1 Concurrency and integrity control datatypes

```
(1)     type activity_class is (UNPROTECTED, PROTECTED, TRANSACTION);
(2)     --  Pcte_activity.activity_class corresponds to the PCTE datatype Activity_class.
```

16.2 Concurrency and integrity control operations

```
        -- 16.2.1 ACTIVITY_ABORT
(1)     procedure abort_activity (
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);
(2)     --  The name of the procedure is modified because 'abort' is an Ada reserved word.

        -- 16.2.2 ACTIVITY_END
(3)     procedure end_activity (
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);
(4)     --  The name of the procedure is modified because 'end' is an Ada reserved word.
```

```
        -- 16.2.3 ACTIVITY_START
(5)     procedure start_activity (
            activity_class : in     Pcte_activity.activity_class;
            status         : in     Pcte_error.handle := EXCEPTION_ONLY);
(6)     -- The name of the procedure is modified to retain the analogy with Pcte_activity.
        -- end_activity.
(7)     package lock is
(8)         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);
(9)         -- Pcte_activity.lock.lock_set_mode corresponds to the PCTE datatype Lock_set_mode.
(10)        subtype internal_mode is Pcte_activity.lock.lock_set_mode
                range READ_UNPROTECTED .. WRITE_PROTECTED;
(11)        -- Pcte_activity.lock.internal_mode corresponds to the PCTE datatype
            -- Lock_internal_mode.

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

            -- 16.2.5 LOCK_SET_INTERNAL_MODE
(13)        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
(14)        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
(15)        procedure unset_object (
                object : in     Pcte.object_reference;
                scope  : in     Pcte.object_scope;
                status : in     Pcte_error.handle := EXCEPTION_ONLY);
(16)    end lock;
```

```
(17)    end Pcte_activity;
```

17 Replication

```
(1)     with Pcte, Pcte_error;
(2)     package Pcte_replica_set is
(3)         use Pcte_error;
```

17.1 Replication datatypes

```
(1)         -- None.
```

17.2 Replication operations

```
            -- 17.2.1 REPLICA_SET_ADD_COPY_VOLUME
(1)         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
(2)         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
(3)         procedure remove (
                replica_set     : in    Pcte.object_reference;
                status          : in    Pcte_error.handle := EXCEPTION_ONLY);

            -- 17.2.4 REPLICA_SET_REMOVE_COPY_VOLUME
(4)         procedure remove_copy_volume (
                replica_set     : in    Pcte.object_reference;
                copy_volume     : in    Pcte.object_reference;
                status          : in    Pcte_error.handle := EXCEPTION_ONLY);
(5)     end Pcte_replica_set;

(6)     with Pcte, Pcte_error;
(7)     package Pcte_replicated_object is
(8)         use Pcte_error;
```

```
        -- 17.2.5 REPLICATED_OBJECT_CREATE
(9)     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
(10)    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
(11)    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
(12)    procedure remove (
            object  : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 17.2.9 WORKSTATION_SELECT_REPLICA_VOLUME
(13)    --  See 18.2.

        -- 17.2.10 WORKSTATION_UNSELECT_REPLICA_VOLUME
(14)    --  See 18.2.
(15) end Pcte_replicated_object;
```

18 Network connection

```
(1)  with Pcte, Pcte_error, Pcte_device, Pcte_discretionary, Pcte_volume;
(2)  package Pcte_workstation is
(3)      use Pcte, Pcte_error, Pcte.reference;
```

18.1 Network connection datatypes

```
(1)      type connection_status is (LOCAL, CLIENT, CONNECTED, AVAILABLE);
(2)      --  Pcte_workstation.connection_status corresponds to the PCTE datatype Connection_status.
(3)      subtype requested_connection_status is Pcte_workstation.connection_status
             range LOCAL .. CONNECTED;
```

(4) -- Pcte_workstation.requested_connection_status corresponds to the PCTE datatype
-- Requested_connection_status.

(5) **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);

(6) -- Pcte_workstation.work_status_item corresponds to the PCTE datatype Work_status_item.

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

(8) -- Pcte_workstation.work_status corresponds to the PCTE datatype Work_status_item.

(9) **type** workstation_status **is record**
connection : Pcte_workstation.connection_status;
work : Pcte_workstation.work_status;
end record;

(10) -- Pcte_workstation.workstation_status corresponds to the PCTE datatype
-- Workstation_status.

18.2 Network connection operations

-- 18.2.1 WORKSTATION_CONNECT

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

-- 18.2.2 WORKSTATION_CREATE

(2) -- 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.

(3) **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);

```
(4)     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

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

        -- 18.2.4 WORKSTATION_DISCONNECT

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

        -- 18.2.5 WORKSTATION_GET_STATUS

(7)     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

(8)     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

(9)     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

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

(11)    end Pcte_workstation;
```

18.3 Foreign system operations

```
    -- 18.3.1 CONTENTS_COPY_FROM_FOREIGN_SYSTEM
(1) --  See 12.2.
    -- 18.3.2 CONTENTS_COPY_TO_FOREIGN_SYSTEM
(2) --  See 12.2.
```

18.4 Time operations

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

19 Discretionary security

```
(1) with Pcte, Pcte_error;
(2) package Pcte_discretionary is
(3)     use Pcte_error;
```

19.1 Discretionary security datatypes

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

```
(7)   -- Pcte_discretionary.object.access_mode_value corresponds to the PCTE datatype
      -- Discretionary_access_mode_value.

(8)   subtype atomic_access_mode_value is Pcte_discretionary.object.mode_value
          range GRANTED .. DENIED;

(9)   -- Pcte_discretionary.object.atomic_access_mode_value corresponds to the PCTE
      -- datatype Atomic_discretionary_access_mode_value.

(10)  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);

(11)  -- Pcte_discretionary.object.access_mode corresponds to the PCTE datatype
      -- Discretionary_access_mode.

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

(13)  -- Pcte_discretionary.object.access_modes corresponds to the PCTE datatype
      -- Discretionary_access_modes.

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

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

(16)  -- 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.

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

(18)  -- 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.

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

(20)  -- 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.


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

(22)  package acl_entries is
```

```
(23)  type sequence is limited private;
(24)  -- Pcte_discretionary.object.acl_entries.sequence corresponds to the PCTE datatype
      -- Acl.
(25)  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;
(26)  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);
(27)  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);
(28)  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);
(29)  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);
(30)  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);
(31)  function length_of (
          list        : Pcte_discretionary.object.acl_entries.sequence;
          status      : Pcte_error.handle := EXCEPTION_ONLY)
          return      Pcte.natural;
(32)  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;
```

```
(33)        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;
(34)        procedure normalize (
                list        : in out    Pcte_discretionary.object.acl_entries.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(35)        procedure discard (
                list        : in out    Pcte_discretionary.object.acl_entries.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(36)    private
(37)        implementation-defined
(38)    end acl_entries;
```

19.2 Discretionary access control operations

```
        -- 19.2.1 GROUP_GET_IDENTIFIER
(1)     --  See 19.3.

        -- 19.2.2 OBJECT_CHECK_PERMISSION
(2)     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
(3)     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
(4)     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);
(5)   end object;
```

19.3 Discretionary security administration operations

```
(1)   package group is

          -- 19.2.1 GROUP_GET_IDENTIFIER

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

          -- 19.3.1 GROUP_INITIALIZE

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

          -- 19.3.2 GROUP_REMOVE

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

          -- 19.3.3 GROUP_RESTORE

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

(6)   end group;


(7)   package program_group is

          -- 19.3.4 PROGRAM_GROUP_ADD_MEMBER

(8)       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

(9)       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

(10)      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
(11)        procedure remove_subgroup (
                group    : in    Pcte.object_reference;
                subgroup : in    Pcte.object_reference;
                status   : in    Pcte_error.handle := EXCEPTION_ONLY);
(12)    end program_group;

(13)    package user_group is

            -- 19.3.8 USER_GROUP_ADD_MEMBER
(14)        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
(15)        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
(16)        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
(17)        procedure remove_subgroup (
                group    : in    Pcte.object_reference;
                subgroup : in    Pcte.object_reference;
                status   : in    Pcte_error.handle := EXCEPTION_ONLY);
(18)    end user_group;
(19) end Pcte_discretionary;
```

20 Mandatory security

```
(1)  with Pcte, Pcte_error, Pcte_discretionary;
(2)  package Pcte_mandatory is
(3)      use Pcte_error;
```

20.1 Mandatory security datatypes

```
(1)      type security_label is new Pcte.string;
```

(2) -- Pcte_mandatory.security_label corresponds to the PCTE datatype Security_label.

(3) **type** floating_level **is** (NO_FLOAT, FLOAT_IN, FLOAT_OUT, FLOAT_IN_OUT);

(4) -- Pcte_mandatory.floating_level corresponds to the PCTE datatype Floating_level.

20.2 Mandatory security operations

(1) **package** device **is**

-- 20.2.1 DEVICE_SET_CONFIDENTIALITY_RANGE

(2) **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

(3) **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);

(4) **end** device;

(5) **package** execution_site **is**

-- 20.2.3 EXECUTION_SITE_SET_CONFIDENTIALITY_RANGE

(6) **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

(7) **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);

(8) **end** execution_site;

(9) **package** object **is**

```
        -- 20.2.5 OBJECT_SET_CONFIDENTIALITY_LABEL
(10)    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
(11)    procedure set_integrity_label (
            object : in  Pcte.object_reference;
            label  : in  Pcte_mandatory.security_label;
            status : in  Pcte_error.handle := EXCEPTION_ONLY);
(12)  end object;

(13)  package volume is

        -- 20.2.7 VOLUME_SET_CONFIDENTIALITY_RANGE
(14)    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
(15)    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);
(16)  end volume;
```

20.3 Mandatory security administration operations

```
(1)   package confidentiality_class is

        -- 20.3.1 CONFIDENTIALITY_CLASS_INITIALIZE
(2)     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);
(3)     procedure initialize (
            object     : in  Pcte.object_reference;
            class_name : in  Pcte.name;
            status     : in  Pcte_error.handle := EXCEPTION_ONLY);
```

```
(4)   end confidentiality_class;

(5)   package group is

        -- 20.3.2 GROUP_DISABLE_FOR_CONFIDENTIALITY_DOWNGRADE

(6)     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

(7)     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

(8)     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

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

(10)  end group;

(11)  package integrity_class is

        -- 20.3.6 INTEGRITY_CLASS_INITIALIZE

(12)    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);

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

(14)  end integrity_class;

(15)  package user is
```

```
        -- 20.3.7 USER_EXTEND_CONFIDENTIALITY_CLEARANCE
(16)    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

(17)    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

(18)    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

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

(20)    end user;

(21) end Pcte_mandatory;
```

20.4 Mandatory security operations for processes

```
(1)  -- See 13.3.
```

21 Auditing

```
(1)  with Pcte, Pcte_error, Pcte_discretionary, Pcte_mandatory;

(2)  package Pcte_audit is

(3)      use Pcte_error;
```

21.1 Auditing datatypes

(1) **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);

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

(3) **subtype** selectable_event_type **is** Pcte_audit.event_type **range** WRITE .. INFORMATION;

(4) -- Pcte_audit.selectable_event_type corresponds to the PCTE datatype
-- Selectable_event_type.

(5) **subtype** mandatory_event_type **is** Pcte_audit.event_type
range CHANGE_IDENTIFICATION .. SECURITY_ADMINISTRATION;

(6) -- Pcte_audit.mandatory_event_type corresponds to the PCTE datatype
-- Mandatory_event_type.

(7) **type** selected_return_code **is** (FAILURE, SUCCESS, ANY_CODE);

(8) -- Pcte_audit.selected_return_code corresponds to the PCTE datatype
-- Selected_return_code.

(9) **subtype** return_code **is** Pcte_audit.selected_return_code **range** FAILURE .. SUCCESS;

(10) -- Pcte_audit.return_code corresponds to the PCTE datatype Return_code.

(11) **type** audit_status **is** (ENABLED, DISABLED);

(12) -- Pcte_audit.audit_status corresponds to the PCTE datatype Audit_status.

(13) **type** general_criterion **is record**
selectable_event_type : Pcte_audit.selectable_event_type;
return_code : Pcte_audit.selected_return_code;
end record;

(14) -- Pcte_audit.general_criterion corresponds to the PCTE datatype General_criterion.

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

(16) **package** general_criteria **is**

(17) **type** sequence **is limited private**;

(18) -- Pcte_audit.general_criteria.sequence corresponds to the PCTE datatype
-- General_criteria.

```
(19)    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;

(20)    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);

(21)    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);

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

(23)    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);

(24)    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);

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

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

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

```
(28)        procedure normalize (
                list        : in out    Pcte_audit.general_criteria.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(29)        procedure discard (
                list        : in out    Pcte_audit.general_criteria.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(30)    private
(31)        implementation-defined
(32)    end general_criteria;


(33)    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;
(34)    -- Pcte_audit.confidentiality_criterion corresponds to the PCTE datatype
        -- Confidentiality_criterion.


(35)    -- The semantics of the operations of this package are defined in 8.2.8.
(36)    package confidentiality_criteria is
(37)        type sequence is limited private;
(38)        -- Pcte_audit.confidentiality_criteria.sequence corresponds to the
            -- PCTE datatype Confidentiality_criteria.
(39)        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;
(40)        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);
(41)        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);
```

```
(42)    procedure append (
            list    : in out    Pcte_audit.confidentiality_criteria.sequence;
            item    : in        Pcte_audit.confidentiality_criterion;
            status  : in        Pcte_error.handle := EXCEPTION_ONLY);
(43)    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);
(44)    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);
(45)    function length_of (
            list    : Pcte_audit.confidentiality_criteria.sequence;
            status  : Pcte_error.handle := EXCEPTION_ONLY)
            return    Pcte.natural;
(46)    function index_of (
            list    : Pcte_audit.confidentiality_criteria.sequence;
            item    : Pcte_audit.confidentiality_criterion;
            status  : Pcte_error.handle := EXCEPTION_ONLY)
            return    Pcte.integer;
(47)    function are_equal (
            first   : Pcte_audit.confidentiality_criteria.sequence;
            second  : Pcte_audit.confidentiality_criteria.sequence;
            status  : Pcte_error.handle := EXCEPTION_ONLY)
            return    Pcte.boolean;
(48)    procedure normalize (
            list    : in out    Pcte_audit.confidentiality_criteria.sequence;
            status  : in        Pcte_error.handle := EXCEPTION_ONLY);
(49)    procedure discard (
            list    : in out    Pcte_audit.confidentiality_criteria.sequence;
            status  : in        Pcte_error.handle := EXCEPTION_ONLY);
(50)  private
(51)    implementation-defined
(52)  end confidentiality_criteria;
```

```
(53)  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;

(54)  -- Pcte_audit.integrity_criterion corresponds to the PCTE datatype Integrity_criterion.

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

(56)  package integrity_criteria is

(57)      type sequence is limited private;

(58)      -- Pcte_audit.integrity_criteria.sequence corresponds to the PCTE datatype
          -- Integrity_criteria.

(59)      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;

(60)      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);

(61)      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);

(62)      procedure append (
              list        : in out    Pcte_audit.integrity_criteria.sequence;
              item        : in        Pcte_audit.integrity_criterion;
              status      : in        Pcte_error.handle := EXCEPTION_ONLY);

(63)      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);

(64)      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);
```

```
(65)        function length_of (
                list        : Pcte_audit.integrity_criteria.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.natural;
(66)        function index_of (
                list        : Pcte_audit.integrity_criteria.sequence;
                item        : Pcte_audit.integrity_criterion;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.integer;
(67)        function are_equal (
                first       : Pcte_audit.integrity_criteria.sequence;
                second      : Pcte_audit.integrity_criteria.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.boolean;
(68)        procedure normalize (
                list        : in out    Pcte_audit.integrity_criteria.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(69)        procedure discard (
                list        : in out    Pcte_audit.integrity_criteria.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(70)    private
(71)        implementation-defined
(72)    end integrity_criteria;


(73)    type object_criterion is record
            selectable_event_type   : Pcte_audit.selectable_event_type;
            object_reference        : Pcte.object_reference;
        end record;
(74)    -- Pcte_audit.object_criterion corresponds to the PCTE datatype Object_criterion.

(75)    -- The semantics of the operations of this package are defined in 8.2.8.
(76)    package object_criteria is
(77)        type sequence is limited private;
(78)        -- Pcte_audit.object_criteria.sequence corresponds to the
            -- PCTE datatype Object_criteria.
(79)        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;
```

```
(80)    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);
(81)    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);
(82)    procedure append (
            list        : in out    Pcte_audit.object_criteria.sequence;
            item        : in        Pcte_audit.object_criterion;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(83)    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);
(84)    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);
(85)    function length_of (
            list        : Pcte_audit.object_criteria.sequence;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.natural;
(86)    function index_of (
            list        : Pcte_audit.object_criteria.sequence;
            item        : Pcte_audit.object_criterion;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.integer;
(87)    function are_equal (
            first       : Pcte_audit.object_criteria.sequence;
            second      : Pcte_audit.object_criteria.sequence;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.boolean;
(88)    procedure normalize (
            list        : in out    Pcte_audit.object_criteria.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
(89)        procedure discard (
                list        : in out    Pcte_audit.object_criteria.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(90)    private
(91)        implementation-defined
(92)    end object_criteria;

(93)    type user_criterion is record
            selectable_event_type   : Pcte_audit.selectable_event_type;
            group                   : Pcte_discretionary.group_identifier;
        end record;
(94)    -- Pcte_audit.user_criterion corresponds to the PCTE datatype User_criterion.

(95)    -- The semantics of the operations of this package are defined in 8.2.8.
(96)    package user_criteria is
(97)        type sequence is limited private;
(98)        -- Pcte_audit.user_criteria.sequence corresponds to the PCTE datatype User_criteria.
(99)        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;
(100)       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);
(101)       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);
(102)       procedure append (
                list    : in out    Pcte_audit.user_criteria.sequence;
                item    : in        Pcte_audit.user_criterion;
                status  : in        Pcte_error.handle := EXCEPTION_ONLY);
(103)       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);
```

```
(104)       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);
(105)       function length_of (
                list          : Pcte_audit.user_criteria.sequence;
                status        : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.natural;
(106)       function index_of (
                list          : Pcte_audit.user_criteria.sequence;
                item          : Pcte_audit.user_criterion;
                status        : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.integer;
(107)       function are_equal (
                first         : Pcte_audit.user_criteria.sequence;
                second        : Pcte_audit.user_criteria.sequence;
                status        : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.boolean;
(108)       procedure normalize (
                list      : in out   Pcte_audit.user_criteria.sequence;
                status    : in       Pcte_error.handle := EXCEPTION_ONLY);
(109)       procedure discard (
                list      : in out   Pcte_audit.user_criteria.sequence;
                status    : in       Pcte_error.handle := EXCEPTION_ONLY);
(110)   private
(111)       implementation-defined
(112)   end user_criteria;
```

21.2 Auditing operations

```
        -- 21.2.1 AUDIT_ADD_CRITERION
(1)     -- AUDIT_ADD_CRITERION is mapped to the overloaded procedures
        -- Pcte_audit.add_criterion according to the type of the parameter criterion.
(2)     procedure add_criterion (
            station     : in     Pcte.object_reference;
            criterion   : in     Pcte_audit.general_criterion;
            status      : in     Pcte_error.handle := EXCEPTION_ONLY);
(3)     procedure add_criterion (
            station     : in     Pcte.object_reference;
            criterion   : in     Pcte_audit.confidentiality_criterion;
            status      : in     Pcte_error.handle := EXCEPTION_ONLY);
```

```
(4)     procedure add_criterion (
            station     : in    Pcte.object_reference;
            criterion   : in    Pcte_audit.integrity_criterion;
            status      : in    Pcte_error.handle := EXCEPTION_ONLY);

(5)     procedure add_criterion (
            station     : in    Pcte.object_reference;
            criterion   : in    Pcte_audit.object_criterion;
            status      : in    Pcte_error.handle := EXCEPTION_ONLY);

(6)     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

(7)     --  See below.

        -- 21.2.3 AUDIT_FILE_READ

(8)     --  See below.

        -- 21.2.4 AUDIT_GET_CRITERIA

(9)     --  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.

(10)    procedure get_criteria (
            station     : in        Pcte.object_reference;
            criteria    : in out    Pcte_audit.general_criteria.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);

(11)    procedure get_criteria (
            station     : in        Pcte.object_reference;
            criteria    : in out    Pcte_audit.confidentiality_criteria.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);

(12)    procedure get_criteria (
            station     : in        Pcte.object_reference;
            criteria    : in out    Pcte_audit.integrity_criteria.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);

(13)    procedure get_criteria (
            station     : in        Pcte.object_reference;
            criteria    : in out    Pcte_audit.object_criteria.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);

(14)    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
(15) -- See below.

    -- 21.2.6 AUDIT_REMOVE_CRITERION
(16) -- AUDIT_REMOVE_CRITERION is mapped to the overloaded procedures
     -- Pcte_audit.remove_criterion according to the type of the parameter criterion.

(17) procedure remove_criterion (
         station   : in  Pcte.object_reference;
         criterion : in  Pcte_audit.selectable_event_type;
         status    : in  Pcte_error.handle := EXCEPTION_ONLY);

(18) procedure remove_criterion (
         station   : in  Pcte.object_reference;
         criterion : in  Pcte_audit.confidentiality_criterion;
         status    : in  Pcte_error.handle := EXCEPTION_ONLY);

(19) procedure remove_criterion (
         station   : in  Pcte.object_reference;
         criterion : in  Pcte_audit.integrity_criterion;
         status    : in  Pcte_error.handle := EXCEPTION_ONLY);

(20) procedure remove_criterion (
         station   : in  Pcte.object_reference;
         criterion : in  Pcte_audit.object_criterion;
         status    : in  Pcte_error.handle := EXCEPTION_ONLY);

(21) procedure remove_criterion (
         station   : in  Pcte.object_reference;
         criterion : in  Pcte_audit.user_criterion;
         status    : in  Pcte_error.handle := EXCEPTION_ONLY);
```

```
(22)  package file is
(23)      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;
(24)      -- 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.

(25)      -- The semantics of the operations of this package are defined in 8.2.8.
(26)  package auditing_records is
(27)      type sequence is limited private;
(28)      -- Pcte_audit.file.auditing_records.sequence corresponds to the PCTE datatype
          -- Audit_file.
(29)      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;
```

```
(30)    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);
(31)    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);
(32)    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);
(33)    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);
(34)    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);
(35)    function length_of (
            list      : Pcte_audit.file.auditing_records.sequence;
            status    : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.natural;
(36)    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;
(37)    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;
(38)    procedure normalize (
            list      : in out   Pcte_audit.file.auditing_records.sequence;
            status    : in       Pcte_error.handle := EXCEPTION_ONLY);
```

```
(39)        procedure discard (
                list        : in out    Pcte_audit.file.auditing_records.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(40)    private
(41)        implementation-defined
(42)    end auditing_records;


        -- 21.2.2 AUDIT_FILE_COPY_AND_RESET
(43)    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
(44)    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);
(45)    -- 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
(46)    procedure write (
            text    : in    Pcte.string;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);
(47)    end file;

        -- 21.2.7 AUDIT_SELECTION_CLEAR
(48)    procedure clear_selection (
            station : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 21.2.8 AUDIT_SWITCH_OFF_SELECTION
(49)    procedure switch_off_selection (
            station : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
        -- 21.2.9 AUDIT_SWITCH_ON_SELECTION
(50)    procedure switch_on_selection (
            station : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);

        -- 21.2.10 AUDITING_STATUS
(51)    function get_status (
            station     : Pcte.object_reference;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte_audit.audit_status;
(52)    end Pcte_audit;
```

22 Accounting

```
(1)     with Pcte, Pcte_error, Pcte_discretionary;
(2)     package Pcte_accounting is
(3)         use Pcte_error;
```

22.1 Accounting datatypes

```
(1)         type consumer_identifier is new Pcte.natural;
(2)         -- Pcte_accounting.consumer_identifier corresponds to the PCTE datatype
            -- Consumer_identifier.
(3)         type resource_identifier is new Pcte.natural;
(4)         -- Pcte_accounting.resource_identifier corresponds to the PCTE datatype
            -- Resource_identifier.
(5)         package log is
(6)             type operation is (SEND, RECEIVE);
(7)             type resource_kind is (WORKSTATION, STATIC_CONTEXT, SDS, DEVICE, FILE,
                    PIPE, MESSAGE_QUEUE, INFORMATION);
```

(8)
```
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;
```

(9)
```
-- 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.
```

(10)
```
-- The semantics of the operations of this package are defined in 8.2.8.
```

(11)
```
package accounting_records is
```

(12)
```
    type sequence is limited private;
```

(13)
```
    -- Pcte_accounting.log.accounting_records.sequence corresponds to the PCTE
    -- datatype Accounting_log.
```

(14)
```
    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;
```

```
(15)    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);
(16)    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);
(17)    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);
(18)    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);
(19)    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);
(20)    function length_of (
            list        : Pcte_accounting.log.accounting_records.sequence;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.natural;
(21)    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;
(22)    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;
(23)    procedure normalize (
            list        : in out    Pcte_accounting.log.accounting_records.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
(24)        procedure discard (
                list        : in out    Pcte_accounting.log.accounting_records.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(25)    private
(26)        implementation-defined
(27)    end accounting_records;
```

22.2 Accounting operations

```
        -- 22.2.1 ACCOUNTING_LOG_COPY_AND_RESET
(1)     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
(2)     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);
(3)     -- 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
(4)     -- See below.

        -- 22.2.4 ACCOUNTING_ON
(5)     -- See below.

        -- 22.2.5 ACCOUNTING_RECORD_WRITE
(6)     procedure write (
            log             : in    Pcte.object_reference;
            information     : in    Pcte.string;
            status          : in    Pcte_error.handle := EXCEPTION_ONLY);
(7)  end log;

        -- 22.2.3 ACCOUNTING_OFF
(8)     procedure off (
            station : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);
```

```
        -- 22.2.4 ACCOUNTING_ON
(9)     procedure on (
            log     : in    Pcte.object_reference;
            station : in    Pcte.object_reference;
            status  : in    Pcte_error.handle := EXCEPTION_ONLY);
(10)    package consumer_group is

            -- 22.2.6 CONSUMER_GROUP_INITIALIZE

(11)        function initialize (
                group       : Pcte.object_reference;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte_accounting.consumer_identifier;

            -- 22.2.7 CONSUMER_GROUP_REMOVE

(12)        procedure remove (
                group  : in     Pcte.object_reference;
                status  : in    Pcte_error.handle := EXCEPTION_ONLY);
(13)    end consumer_group;
(14)    package resource_group is

            -- 22.2.8 RESOURCE_GROUP_ADD_OBJECT

(15)        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

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

            -- 22.2.10 RESOURCE_GROUP_INITIALIZE

(17)        function initialize (
                group       : Pcte.object_reference;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte_accounting.resource_identifier;

            -- 22.2.11 RESOURCE_GROUP_REMOVE_OBJECT

(18)        procedure remove_object (
                object : in     Pcte.object_reference;
                group  : in     Pcte.object_reference;
                status  : in    Pcte_error.handle := EXCEPTION_ONLY);
(19)    end resource_group;
(20)end Pcte_accounting;
```

22.3 Consumer identity operations

(1) -- See 13.3.

23 References

(1) -- See 8.4.

24 Limits

(1) **with** Pcte, Pcte_error;

(2) **package** Pcte_limit **is**

(3) **use** Pcte_error;

(4) **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);

(5) **type** category **is** (STANDARD_LIMIT, IMPLEMENTATION_LIMIT,
REMAINING_LIMIT);

```
(6)     -- Pcte_limit.category indicates to the procedure get_value the category of limit value
        -- required.  STANDARD_LIMIT indicates the limit bounds given in ECMA-149, clause
        -- 24; IMPLEMENTATION_LIMIT indicates the limits imposed by the implementation;
        -- and REMAINING_LIMIT indicates the current available quantity of the resource
        -- concerned.

(7)     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);

(8)     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);

(9)     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);

(10)    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);

(11)    -- 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.

(12)  end Pcte_limit;
```

25 Errors

```
(1)   package Pcte_error is

(2)       type handle is limited private;
```

```
(3)     -- 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.
(4)     EXCEPTION_ONLY : constant Pcte_error.handle;
(5)     -- EXCEPTION_ONLY is a value of an error handle which has raise set, record unset, and
        -- has recorded error condition NO_ERROR.

        ---------------------------------  PCTE exceptions  --------------------------------

(6)     -- The error conditions defined in ECMA-149, 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.
(7)     ERROR_WITH_USAGE : exception;                                          -- U
(8)     -- indicates that the intended usage of PCTE is an error.
(9)     ERROR_WITH_STATE : exception;                                          -- S
(10)    -- indicates that the call is an error in the context of the state of the object base.
(11)    ERROR_WITH_IDENTIFICATION : exception;                                 -- I
(12)    -- indicates that the parameters fail to identify a valid entity of the intended kind.
(13)    ERROR_WITH_PARAMETER : exception;                                      -- P
(14)    -- indicates a fundamental error in the values of one or more parameters, such that the call
        -- could not be valid under any circumstances
(15)    ERROR_WITH_DEVICE : exception;                                         -- D
(16)    -- indicates that there is a problem with the use of a device or a volume.
(17)    ERROR_WITH_NETWORK : exception;                                        -- N
(18)    -- indicates that there is a problem with access over the network.
(19)    ERROR_FROM_INTERRUPTED : exception;                                    -- R
(20)    -- indicates that the operation has been interrupted.
(21)    ERROR_FROM_TIMEOUT : exception;                                        -- T
(22)    -- indicates that the operation has been timed out.
(23)    ERROR_WITH_TIME : exception;
(24)    -- 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  -------------------------------

(25)    type error_code is (
```

(26) NO_ERROR,

-- Errors defined in ECMA-149:
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
ENUMERAL_TYPES_ARE_MULTIPLE,                                   -- P
EVALUATION_STATUS_IS_INCONSISTENT_WITH_EVALUATION_POINT,       -- S
EVENT_TYPE_IS_NOT_SELECTED,                                    -- S
EXECUTION_CLASS_HAS_NO_USABLE_EXECUTION_SITES,                 -- U
EXECUTION_SITE_IS_INACCESSIBLE,                                -- N
EXECUTION_SITE_IS_NOT_IN_EXECUTION_CLASS,                      -- U
EXECUTION_SITE_IS_UNKNOWN,                                     -- S
EXTERNAL_LINK_IS_BAD,                                          -- U
EXTERNAL_LINK_IS_NOT_DUPLICABLE,                               -- U
FOREIGN_DEVICE_IS_INVALID,                                     -- D
FOREIGN_EXECUTION_IMAGE_HAS_NO_SITE,                           -- U
FOREIGN_EXECUTION_IMAGE_IS_BEING_EXECUTED,                     -- U
FOREIGN_OBJECT_IS_INACCESSIBLE,                                -- S
FOREIGN_SYSTEM_IS_INACCESSIBLE,                                -- S
FOREIGN_SYSTEM_IS_INVALID,                                     -- S
FOREIGN_SYSTEM_IS_UNKNOWN,                                     -- I
GROUP_IDENTIFIER_IS_IN_USE,                                    -- U
GROUP_IDENTIFIER_IS_INVALID,                                   -- I
IMAGE_IS_ALREADY_ASSOCIATED,                                   -- U
IMAGE_IS_DUPLICATED,                                           -- S
INTEGRITY_CONFINEMENT_WOULD_BE_VIOLATED,                       -- S
INTEGRITY_CRITERION_IS_NOT_SELECTED,                           -- S
INTEGRITY_LABEL_IS_INVALID,                                    -- P
INTEGRITY_WOULD_BE_VIOLATED,                                   -- S
INTERPRETER_IS_INTERPRETABLE,                                  -- S
INTERPRETER_IS_NOT_AVAILABLE,                                  -- U
KEY_ATTRIBUTE_TYPE_UNAPPLY_IS_FORBIDDEN,                       -- U
KEY_IS_BAD,                                                    -- P
KEY_IS_NOT_SYSTEM_KEY,                                         -- P
KEY_SYNTAX_IS_WRONG,                                           -- P
KEY_TYPE_IS_BAD,                                               -- P
KEY_TYPES_ARE_MULTIPLE,                                        -- P
KEY_UPDATE_IS_FORBIDDEN,                                       -- P
KEY_VALUE_AND_EVALUATION_POINT_ARE_INCONSISTENT,               -- P
KEY_VALUE_DOES_NOT_EXIST,                                      -- I
LABEL_IS_OUTSIDE_RANGE,                                        -- S
LABEL_RANGE_IS_BAD,                                            -- P
LAN_ERROR_EXISTS,                                              -- N
LIMIT_WOULD_BE_EXCEEDED,                                       -- U
LINK_DESTINATION_DOES_NOT_EXIST,                               -- S
LINK_DESTINATION_IS_NOT_VISIBLE,                               -- U
LINK_DOES_NOT_EXIST,                                           -- I
LINK_EXCLUSIVENESS_WOULD_BE_VIOLATED,                          --U
LINK_EXISTS,                                                   -- U
LINK_NAME_IS_TOO_LONG_IN_CURRENT_WORKING_SCHEMA,               -- S
```

```
LINK_NAME_SYNTAX_IS_WRONG,                                  -- P
LINK_REFERENCE_IS_NOT_EVALUATED,                            -- P
LINK_REFERENCE_IS_UNSET,                                    -- U
LINK_TYPE_CATEGORY_IS_BAD,                                  -- P
LINK_TYPE_IS_NOT_APPLIED_TO_OBJECT_TYPE,                    -- U
LINK_TYPE_IS_NOT_VISIBLE,                                   -- S
LINK_TYPE_IS_UNKNOWN,                                       -- I
LINK_TYPE_PROPERTIES_AND_KEY_TYPES_ARE_INCONSISTENT,        -- P
LINK_TYPE_PROPERTIES_ARE_INCONSISTENT,                      -- P
LOCK_COULD_NOT_BE_ESTABLISHED,                              -- U
LOCK_INTERNAL_MODE_CANNOT_BE_CHANGED,                       -- S
LOCK_IS_NOT_EXPLICIT,                                       -- U
LOCK_MODE_IS_NOT_ALLOWED,                                   -- U
LOCK_MODE_IS_TOO_STRONG,                                    -- U
LOWER_BOUND_WOULD_BE_VIOLATED,                              -- U
MANDATORY_CLASS_IS_ALREADY_DOMINATED,                       -- U
MANDATORY_CLASS_IS_KNOWN,                                   -- S
MANDATORY_CLASS_IS_UNKNOWN,                                 -- I
MANDATORY_CLASS_NAME_IS_IN_USE,                             -- S
MAXIMUM_USAGE_MODE_WOULD_BE_EXCEEDED,                       -- P
MEMORY_ADDRESS_IS_OUT_OF_PROCESS,                           -- P
MEMORY_REGION_IS_NOT_IN_PROFILING_SPACE,                    -- U
MESSAGE_IS_NOT_A_NOTIFICATION_MESSAGE                       -- I
MESSAGE_POSITION_IS_NOT_VALID,                              -- I
MESSAGE_QUEUE_HAS_BEEN_DELETED,                             -- U
MESSAGE_QUEUE_HAS_BEEN_WOKEN,                               -- U
MESSAGE_QUEUE_HAS_NO_HANDLER,                               -- U
MESSAGE_QUEUE_IS_BUSY,                                      -- S
MESSAGE_QUEUE_IS_NOT_RESERVED,                              -- U
MESSAGE_QUEUE_IS_RESERVED,                                  -- U
MESSAGE_QUEUE_TOTAL_SPACE_WOULD_BE_TOO_SMALL,               -- S
MESSAGE_QUEUE_WOULD_BE_TOO_BIG,                             -- S
MESSAGE_TYPES_NOT_FOUND_IN_QUEUE,                           -- S
NON_BLOCKING_IO_IS_INVALID,                                 -- U
NOTIFIER_KEY_DOES_NOT_EXIST,                                -- S
NOTIFIER_KEY_EXISTS,                                        -- S
OBJECT_ARCHIVING_IS_INVALID,                                -- U
OBJECT_CANNOT_BE_STABILIZED,                                -- U
OBJECT_CRITERION_IS_NOT_SELECTED,                           -- U
OBJECT_HAS_COPIES,                                          -- U
OBJECT_HAS_EXTERNAL_LINKS_PREVENTING_DELETION,              -- U
OBJECT_HAS_GROUP_WHICH_IS_ALREADY_OWNER,                    -- S
OBJECT_HAS_INTERNAL_LINKS_PREVENTING_DELETION,              -- U
OBJECT_HAS_LINKS_PREVENTING_DELETION,                       -- U
OBJECT_IS_A_PROCESS,                                        -- U
OBJECT_IS_A_REPLICA_SET,                                    -- U
OBJECT_IS_ALREADY_IN_RESOURCE_GROUP,                        -- U
OBJECT_IS_ARCHIVED,                                         -- U
OBJECT_IS_IN_USE_FOR_DELETE,                                -- U
OBJECT_IS_IN_USE_FOR_MOVE,                                  -- U
```

```
OBJECT_IS_INACCESSIBLE,                                          -- U
OBJECT_IS_INACCESSIBLY_ARCHIVED,                                 -- U
OBJECT_IS_LOCKED,                                                -- U
OBJECT_IS_NOT_ACCOUNTABLE_RESOURCE,                              -- U
OBJECT_IS_NOT_ARCHIVED,                                          -- U
OBJECT_IS_NOT_IN_RESOURCE_GROUP,                                 -- U
OBJECT_IS_NOT_LOCKED,                                            -- U
OBJECT_IS_NOT_MASTER_REPLICATED_OBJECT,                          -- U
OBJECT_IS_NOT_MOVABLE,                                           -- U
OBJECT_IS_NOT_ON_ADMINISTRATION_VOLUME,                          -- U
OBJECT_IS_NOT_ON_MASTER_VOLUME_OF_REPLICA_SET,                   -- U
OBJECT_IS_NOT_REPLICABLE,                                        -- U
OBJECT_IS_NOT_REPLICATED_ON_VOLUME,                              -- U
OBJECT_IS_OF_WRONG_TYPE,                                         -- U
OBJECT_IS_OPERATED_ON,                                           -- S
OBJECT_IS_PREDEFINED_REPLICATED,                                 -- U
OBJECT_IS_REPLICATED,                                            -- U
OBJECT_IS_STABLE,                                                -- U
OBJECT_LABEL_CANNOT_BE_CHANGED_IN_TRANSACTION,                   -- S
OBJECT_OWNER_CONSTRAINT_WOULD_BE_VIOLATED,                       -- U
OBJECT_OWNER_VALUE_WOULD_BE_INCONSISTENT_WITH_ATOMIC_ACL,        -- U
OBJECT_REFERENCE_IS_INTERNAL,                                    -- U
OBJECT_REFERENCE_IS_INVALID,                                     -- U
OBJECT_REFERENCE_IS_UNSET,                                       -- U
OBJECT_TYPE_IS_ALREADY_IN_DESTINATION_SET,                       -- U
OBJECT_TYPE_IS_INVALID,                                          -- P
OBJECT_TYPE_IS_NOT_IN_DESTINATION_SET,                           -- U
OBJECT_TYPE_IS_NOT_VISIBLE,                                      -- S
OBJECT_TYPE_IS_UNKNOWN,                                          -- I
OBJECT_TYPE_WOULD_HAVE_NO_PARENT_TYPE,                           -- P
OBJECT_TYPES_MISMATCH,                                           -- P
OPEN_KEY_IS_INVALID,                                             -- S
OPENING_MODE_IS_INVALID,                                         -- P
OPERATION_HAS_TIMED_OUT,                                         -- T
OPERATION_IS_INTERRUPTED,                                        -- R
OPERATION_IS_NOT_ALLOWED_ON_TYPE,                                -- U
PARENT_BASIC_TYPES_ARE_MULTIPLE,                                 -- P
PATHNAME_SYNTAX_IS_WRONG,                                        -- I
POSITION_HANDLE_IS_INVALID,                                      -- I
POSITION_IS_INVALID,                                             -- P
POSITIONING_IS_INVALID,                                          -- I
PREFERENCE_DOES_NOT_EXIST,                                       -- I
PREFERRED_LINK_KEY_IS_BAD,                                       -- S
PREFERRED_LINK_TYPE_IS_UNSET,                                    -- S
PRIVILEGE_IS_NOT_GRANTED,                                        -- I
PROCESS_CONFIDENTIALITY_IS_NOT_DOMINATED,                        -- S
PROCESS_HAS_NO_UNTERMINATED_CHILD,                               -- U
PROCESS_INTEGRITY_DOES_NOT_DOMINATE,                             -- S
PROCESS_IS_IN_TRANSACTION,                                       -- U
PROCESS_IS_INACCESSIBLE,                                         -- U
```

```
PROCESS_IS_INITIAL_PROCESS,                                -- U
PROCESS_IS_NOT_ANCESTOR,                                   -- U
PROCESS_IS_NOT_CHILD,                                      -- U
PROCESS_IS_NOT_TERMINABLE_CHILD,                           -- U
PROCESS_IS_NOT_THE_CALLER,                                 -- U
PROCESS_IS_THE_CALLER,                                     -- U
PROCESS_IS_UNKNOWN,                                        -- I
PROCESS_LABELS_WOULD_BE_INCOMPATIBLE,                      -- S
PROCESS_LACKS_REQUIRED_STATUS,                             -- S
PROCESS_TERMINATION_IS_ALREADY_ACKNOWLEDGED,               -- S
PROFILING_IS_NOT_SWITCHED_ON,                              -- S
PROGRAM_GROUP_IS_NOT_EMPTY,                                -- S
RANGE_IS_OUTSIDE_RANGE,                                    -- S
REFERENCE_CANNOT_BE_ALLOCATED,                             -- S
REFERENCE_NAME_IS_INVALID,                                 -- I
REFERENCED_OBJECT_IS_NOT_MUTABLE,                          -- U
REFERENCED_OBJECT_IS_UNSET,                                -- U
RELATIONSHIP_TYPE_PROPERTIES_ARE_INCONSISTENT,             -- P
REPLICA_SET_COPY_IS_NOT_EMPTY,                             -- S
REPLICA_SET_HAS_COPY_VOLUMES,                              -- S
REPLICA_SET_IS_NOT_EMPTY,                                  -- S
REPLICA_SET_IS_NOT_KNOWN,                                  -- I
REPLICATED_COPY_IS_IN_USE,                                 -- S
REPLICATED_COPY_UPDATE_IS_FORBIDDEN,                       -- U
RESOURCE_GROUP_IS_KNOWN,                                   -- U
RESOURCE_GROUP_IS_UNKNOWN,                                 -- I
REVERSE_KEY_IS_BAD,                                        -- U
REVERSE_KEY_IS_NOT_SUPPLIED,                               -- U
REVERSE_KEY_IS_SUPPLIED,                                   -- P
REVERSE_LINK_EXISTS,                                       -- U
SDS_IS_IN_A_WORKING_SCHEMA,                                -- U
SDS_IS_KNOWN,                                              -- U
SDS_IS_NOT_EMPTY_NOR_VERSION,                              -- U
SDS_IS_UNDER_MODIFICATION,                                 -- U
SDS_IS_UNKNOWN,                                            -- I
SDS_NAME_IS_DUPLICATE,                                     -- U
SDS_NAME_IS_INVALID,                                       -- P
SDS_WOULD_APPEAR_TWICE_IN_WORKING_SCHEMA,                  -- U
SECURITY_GROUP_ALREADY_HAS_THIS_SUBGROUP,                  -- U
SECURITY_GROUP_IS_ALREADY_ENABLED,                         -- U
SECURITY_GROUP_IS_IN_USE,                                  -- U
SECURITY_GROUP_IS_KNOWN,                                   -- U
SECURITY_GROUP_IS_NOT_A_SUBGROUP,                          -- U
SECURITY_GROUP_IS_NOT_ADOPTABLE,                           -- U
SECURITY_GROUP_IS_NOT_ENABLED,                             -- U
SECURITY_GROUP_IS_PREDEFINED,                              -- U
SECURITY_GROUP_IS_REQUIRED_BY_OTHER_GROUPS,                -- U
SECURITY_GROUP_IS_UNKNOWN,                                 -- I
SECURITY_GROUP_WOULD_BE_IN_INVALID_GRAPH,                  -- U
SECURITY_POLICY_WOULD_BE_VIOLATED,                         -- U
```

```
STATIC_CONTEXT_CONTENTS_CANNOT_BE_EXECUTED,                    -- U
STATIC_CONTEXT_IS_ALREADY_MEMBER,                              -- U
STATIC_CONTEXT_IS_BEING_WRITTEN,                               -- S
STATIC_CONTEXT_IS_IN_USE,                                      -- S
STATIC_CONTEXT_IS_NOT_MEMBER,                                  -- U
STATIC_CONTEXT_REQUIRES_TOO_MUCH_MEMORY,                       -- S
STATUS_IS_BAD,                                                 -- S
TIME_CANNOT_BE_CHANGED,                                        -- U
TRANSACTION_CANNOT_BE_COMMITTED,                               -- U
TYPE_HAS_DEPENDENCIES,                                         -- U
TYPE_HAS_NO_LOCAL_NAME,                                        -- U
TYPE_IDENTIFIER_IS_INVALID,                                    -- I
TYPE_IDENTIFIER_SYNTAX_IS_WRONG,                               -- I
TYPE_IDENTIFIER_USAGE_IS_INVALID,                              -- U
TYPE_IS_ALREADY_APPLIED,                                       -- U
TYPE_IS_ALREADY_KNOWN_IN_SDS,                                  -- U
TYPE_IS_NOT_APPLIED,                                           -- U
TYPE_IS_NOT_DESCENDANT,                                        -- U
TYPE_IS_NOT_VISIBLE,                                           -- U
TYPE_IS_OF_WRONG_KIND,                                         -- I
TYPE_IS_UNKNOWN,                                               -- I
TYPE_IS_UNKNOWN_IN_SDS,                                        -- I
TYPE_IS_UNKNOWN_IN_WORKING_SCHEMA,                             -- I
TYPE_NAME_IN_SDS_IS_DUPLICATE,                                 -- U
TYPE_NAME_IS_INVALID,                                          -- P
TYPE_OF_OBJECT_IS_INVALID,                                     -- U
TYPE_REFERENCE_IS_INVALID,                                     -- U
TYPE_REFERENCE_IS_UNSET,                                       -- U
UNLOCKING_IN_TRANSACTION_IS_FORBIDDEN,                         -- U
UPPER_BOUND_WOULD_BE_VIOLATED,                                 -- U
USAGE_MODE_ON_ATTRIBUTE_TYPE_WOULD_BE_VIOLATED,                -- U
USAGE_MODE_ON_LINK_TYPE_WOULD_BE_VIOLATED,                     -- U
USAGE_MODE_ON_OBJECT_TYPE_WOULD_BE_VIOLATED,                   -- U
USER_CRITERION_IS_NOT_SELECTED,                                -- U
USER_GROUP_IS_IN_USE,                                          -- U
USER_GROUP_LACKS_ALL_USERS_AS_SUPERGROUP,                      -- U
USER_GROUP_WOULD_NOT_HAVE_ALL_USERS_AS_SUPERGROUP,             -- U
USER_IS_ALREADY_CLEARED_TO_CLASS,                              -- U
USER_IS_ALREADY_MEMBER,                                        -- U
USER_IS_IN_USE,                                                -- U
USER_IS_NOT_CLEARED,                                           -- U
USER_IS_NOT_CLEARED_TO_CLASS,                                  -- U
USER_IS_NOT_MEMBER,                                            -- U
USER_IS_UNKNOWN,                                               -- I
VALUE_TYPE_IS_INVALID,                                         -- P
VERSION_GRAPH_IS_INVALID,                                      -- U
VERSION_IS_REQUIRED,                                           -- U
VOLUME_CANNOT_BE_MOUNTED_ON_DEVICE,                            -- D
VOLUME_EXISTS,                                                 -- U
VOLUME_HAS_OBJECT_OUTSIDE_RANGE,                               -- S
```

```
   VOLUME_HAS_OBJECTS_IN_USE,                                    -- D
   VOLUME_HAS_OTHER_LINKS,                                       -- U
   VOLUME_HAS_OTHER_OBJECTS,                                     -- U
   VOLUME_IDENTIFIER_IS_INVALID,                                 -- I
   VOLUME_IS_ADMINISTRATION_VOLUME,                              -- U
   VOLUME_IS_ALREADY_COPY_VOLUME_OF_REPLICA_SET,                 -- D
   VOLUME_IS_ALREADY_MOUNTED,                                    -- D
   VOLUME_IS_FULL,                                               -- D
   VOLUME_IS_INACCESSIBLE,                                       -- D
   VOLUME_IS_MASTER_VOLUME_OF_REPLICA_SET,                       -- D
   VOLUME_IS_NOT_COPY_VOLUME_OF_REPLICA_SET,                     -- D
   VOLUME_IS_NOT_MASTER_OR_COPY_VOLUME_OF_REPLICA_SET,           -- D
   VOLUME_IS_READ_ONLY,                                          -- D
   VOLUME_IS_UNKNOWN,                                            -- I
   WORKSTATION_EXISTS,                                           -- U
   WORKSTATION_HAS_NO_CHOICE_OF_VOLUME_FOR_REPLICA_SET,          -- D
   WORKSTATION_IDENTIFIER_IS_INVALID,                            -- I
   WORKSTATION_IS_BUSY,                                          -- U
   WORKSTATION_IS_CONNECTED,                                     -- U
   WORKSTATION_IS_NOT_CONNECTED,                                 -- U
   WORKSTATION_IS_UNKNOWN,                                       -- I

   -- Ada binding specific errors:

   CONTENTS_HANDLE_IS_OPEN,                                      -- S
   INVALID_INDEX,                                                -- P
   STRING_IS_TOO_SHORT,                                          -- P
   UNDEFINED_REFERENCE),                                         -- I

   -- Errors for fine-grain data

   CLUSTER_EXISTS,                                               -- U
   CLUSTER_HAS_OTHER_LINKS,                                      -- U
   CLUSTER_IS_UNKNOWN,                                           -- I
   OBJECT_CANNOT_BE_CLUSTERED,                                   -- U
   OBJECT_IS_FINE_GRAIN,                                         -- U

   -- Errors for object-orientation

   NUMBER_OF_PARAMETERS_IS_WRONG,                                -- U
   OPERATION_METHOD_NOT_FOUND,                                   -- U
   OPERATION_METHOD_COULD_NOT_BE_ACTIVATED,                      -- U
   TYPE_IS_ALREADY_CONSTRAINED,                                  -- U
   TYPE_OF_PARAMETER_IS_WRONG;                                   -- U
```

(27) -- UNDEFINED_REFERENCE indicates that an attempt has been made to use an object
-- reference which has not been set.

(28) -- CONTENTS_HANDLE_IS_OPEN indicates that the contents handle which has been provided
-- as a parameter to the operation is already open; see 12.2.

```
(29)    -- INVALID_INDEX indicates that an attempt has been made to get a non-existent element from
        -- a sequence.
(30)    -- STRING_IS_TOO_SHORT indicates that  there is not enough space to hold the returned
        -- value.

        ----------------------------        OPERATIONS ON STATUS        ----------------------------
(31)        procedure set (
                status      : in out    Pcte_error.handle;
                to_record   : in        STANDARD.BOOLEAN := TRUE;
                to_raise    : in        STANDARD.BOOLEAN := TRUE);
(32)        -- Pcte_error_set sets the record and raise properties of status to the values of to_record and
            -- to_raise respectively.
(33)        procedure unset (
                status  : in out    Pcte_error.handle);
(34)        -- Pcte_error.unset unsets the record and raise properties of status.
(35)        procedure set_will_raise (
                status      : in out    Pcte_error.handle;
                to_raise    : in        STANDARD.BOOLEAN);
(36)        -- Pcte_error.set_will_raise sets the raise property of status to the value of to_raise.
(37)        procedure set_will_record (
                status      : in out    Pcte_error.handle;
                to_record   : in        STANDARD.BOOLEAN);
(38)        -- Pcte_error.set_will_record sets the record property of status to the value of to_record.
(39)        function will_raise (
                status      : Pcte_error.handle)
                return      STANDARD.BOOLEAN;
(40)        -- Pcte_error.will_raise returns the setting of the raise property of status.
(41)        function will_record (
                status      : Pcte_error.handle)
                return      STANDARD.BOOLEAN;
(42)        -- Pcte_error.will_record returns the setting of the record property of status.
(43)        function last_error (
                status      : Pcte_error.handle)
                return      Pcte_error.error_code;
(44)        -- Pcte_error.last_error returns the latest error code recorded in status.
(45)    private
(46)        implementation-defined
(47)    end Pcte_error;
```

Annex A
(normative)

The object orientation module

(1) This annex defines the Ada language binding of the datatypes and operations of the object orientation module defined in annex G of ECMA-149.

A.1 Object-oriented invocation management (see G.2)

(1) **with** Pcte, Pcte_error;

(2) **package** Pcte_oo **is**

(3) **use** Pcte, Pcte_error;

A.1.1 Object-oriented invocation management datatypes

(1)
```
type parameter_constraint is (
    CONSTRAINED_TO_ATTRIBUTE,
    CONSTRAINED_TO_OBJECT,
    CONSTRAINED_TO_INTERFACE);
```

(2)
```
type parameter_item is record (
    constraint : Pcte_oo.parameter_constraint := CONSTRAINED_TO_ATTRIBUTE)
is record
    case constraint is
        when CONSTRAINED_TO_ATTRIBUTE =>
            value       : Pcte.attribute_value;
        when CONSTRAINED_TO_OBJECT =>
            object      : Pcte.object_reference;
        when CONSTRAINED_TO_INTERFACE =>
            interface   : Pcte.object_reference;
    end case;
end record;
```

(3)
```
-- The semantics of the operations of this package are defined in 8.2.8.
```

(4)
```
package parameter_items is
```

(5)
```
    type sequence is limited private;
```

(6)
```
    -- Pcte_oo.parameter_items.sequence corresponds to the PCTE datatype
    -- Parameter_items.
```

(7)
```
    function get (
        list        : Pcte_oo.parameter_items.sequence;
        index       : Pcte.natural := Pcte.natural'FIRST;
        status      : Pcte_error.handle := EXCEPTION_ONLY)
        return      Pcte.name;
```

```
(8)     procedure insert (
            list        : in out    Pcte_oo.parameter_items.sequence;
            item        : in        Pcte_oo.parameter_item;
            index       : in        Pcte.natural := Pcte.natural'LAST;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);

(9)     procedure replace (
            list        : in out    Pcte_oo.parameter_items.sequence;
            item        : in        Pcte_oo.parameter_item;
            index       : in        Pcte.natural := Pcte.natural'LAST;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);

(10)    procedure append (
            list        : in out    Pcte_oo.parameter_items.sequence;
            item        : in        Pcte_oo.parameter_item;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);

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

(12)    procedure copy (
            into_list       : in out    Pcte_oo.parameter_items.sequence;
            from_list       : in        Pcte_oo.parameter_items.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);

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

(14)    function index_of (
            list        : Pcte_oo.parameter_items.sequence;
            item        : Pcte_oo.parameter_item;
            status      : Pcte_error.handle := EXCEPTION_ONLY)
            return      Pcte.integer;

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

(16)    procedure normalize (
            list        : in out    Pcte_oo.parameter_items.sequence;
            status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
(17)        procedure discard (
                list        : in out    Pcte_oo.parameter_items.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(18)    private
(19)        implementation-defined
(20)    end parameter_items;

(21)    type method_request is record
            target_reference  : Pcte.object_reference;
            operation_id      : Pcte.type_reference;
            parameters        : Pcte_oo.parameter_items.sequence;
            context           : Pcte.object_reference;
        end record;

(22)    -- The semantics of the operations of this package are defined in 8.2.8.
(23)    package method_requests is
(24)        type sequence is limited private;
(25)        -- Pcte_oo.method_requests.sequence corresponds to the PCTE datatype
            -- Method_requests.
(26)        function get (
                list        : Pcte_oo.method_requests.sequence;
                index       : Pcte.natural := Pcte.natural'FIRST;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.name;
(27)        procedure insert (
                list        : in out    Pcte_oo.method_requests.sequence;
                item        : in        Pcte_oo.method_request;
                index       : in        Pcte.natural := Pcte.natural'LAST;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(28)        procedure replace (
                list        : in out    Pcte_oo.method_requests.sequence;
                item        : in        Pcte_oo.method_request;
                index       : in        Pcte.natural := Pcte.natural'LAST;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(29)        procedure append (
                list        : in out    Pcte_oo.method_requests.sequence;
                item        : in        Pcte_oo.method_request;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
(30)        procedure delete (
                list        : in out    Pcte_oo.method_requests.sequence;
                index       : in        Pcte.natural := Pcte.natural'FIRST;
                count       : in        Pcte.positive := Pcte.positive'LAST;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(31)        procedure copy (
                into_list       : in out    Pcte_oo.method_requests.sequence;
                from_list       : in        Pcte_oo.method_requests.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);
(32)        function length_of (
                list        : Pcte_oo.method_requests.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.natural;
(33)        function index_of (
                list        : Pcte_oo.method_requests.sequence;
                item        : Pcte_oo.method_request;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.integer;
(34)        function are_equal (
                first       : Pcte_oo.method_requests.sequence;
                second      : Pcte_oo.method_requests.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.boolean;
(35)        procedure normalize (
                list        : in out    Pcte_oo.method_requests.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(36)        procedure discard (
                list        : in out    Pcte_oo.method_requests.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(37)    private
(38)        implementation-defined
(39)    end method_requests;


(40)    type context_adoption is (
            PCTE_ADOPT_WORKING_SCHEMA, PCTE_ADOPT_ACTIVITY,
            PCTE_ADOPT_USER, PCTE_ADOPT_OPEN_OBJECTS,
            PCTE_ADOPT_REFERENCE_OBJECTS, PCTE_ADOPT_ALL);


(41)    -- The semantics of the operations of this package are defined in 8.2.8.
(42)    package context_adoptions is
```

```
(43)  type sequence is limited private;

(44)  -- Pcte_oo.context_adoptions.sequence corresponds to the PCTE datatype
      -- Context_adoptions.

(45)  function get (
          list      : Pcte_oo.context_adoptions.sequence;
          index     : Pcte.natural := Pcte.natural'FIRST;
          status    : Pcte_error.handle := EXCEPTION_ONLY)
          return      Pcte.name;

(46)  procedure insert (
          list      : in out  Pcte_oo.context_adoptions.sequence;
          item      : in      Pcte_oo.context_adoption;
          index     : in      Pcte.natural := Pcte.natural'LAST;
          status    : in      Pcte_error.handle := EXCEPTION_ONLY);

(47)  procedure replace (
          list      : in out  Pcte_oo.context_adoptions.sequence;
          item      : in      Pcte_oo.context_adoption;
          index     : in      Pcte.natural := Pcte.natural'LAST;
          status    : in      Pcte_error.handle := EXCEPTION_ONLY);

(48)  procedure append (
          list      : in out  Pcte_oo.context_adoptions.sequence;
          item      : in      Pcte_oo.context_adoption;
          status    : in      Pcte_error.handle := EXCEPTION_ONLY);

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

(50)  procedure copy (
          into_list     : in out  Pcte_oo.context_adoptions.sequence;
          from_list     : in      Pcte_oo.context_adoptions.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);

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

(52)  function index_of (
          list      : Pcte_oo.context_adoptions.sequence;
          item      : Pcte_oo.context_adoption;
          status    : Pcte_error.handle := EXCEPTION_ONLY)
          return      Pcte.integer;
```

```
(53)        function are_equal (
                first       : Pcte_oo.context_adoptions.sequence;
                second      : Pcte_oo.context_adoptions.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.boolean;
(54)        procedure normalize (
                list    : in out    Pcte_oo.context_adoptions.sequence;
                status  : in        Pcte_error.handle := EXCEPTION_ONLY);
(55)        procedure discard (
                list    : in out    Pcte_oo.context_adoptions.sequence;
                status  : in        Pcte_error.handle := EXCEPTION_ONLY);
(56)    private
(57)        implementation-defined
(58)    end context_adoptions;

(59)    type method_request_id is limited private;

(60)    -- The semantics of the operations of this package are defined in 8.2.8.
(61)    package method_request_ids is
(62)        type sequence is limited private;
(63)        -- Pcte_oo.method_request_ids.sequence corresponds to the PCTE datatype
            -- Method_request_ids.
(64)        function get (
                list        : Pcte_oo.method_request_ids.sequence;
                index       : Pcte.natural := Pcte.natural'FIRST;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return        Pcte.name;
(65)        procedure insert (
                list        : in out    Pcte_oo.method_request_ids.sequence;
                item        : in        Pcte_oo.method_request_id;
                index       : in        Pcte.natural := Pcte.natural'LAST;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(66)        procedure replace (
                list        : in out    Pcte_oo.method_request_ids.sequence;
                item        : in        Pcte_oo.method_request_id;
                index       : in        Pcte.natural := Pcte.natural'LAST;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(67)        procedure append (
                list        : in out    Pcte_oo.method_request_ids.sequence;
                item        : in        Pcte_oo.method_request_id;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
```

```
(68)        procedure delete (
                list        : in out    Pcte_oo.method_request_ids.sequence;
                index       : in        Pcte.natural := Pcte.natural'FIRST;
                count       : in        Pcte.positive := Pcte.positive'LAST;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(69)        procedure copy (
                into_list   : in out    Pcte_oo.method_request_ids.sequence;
                from_list   : in        Pcte_oo.method_request_ids.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);
(70)        function length_of (
                list        : Pcte_oo.method_request_ids.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.natural;
(71)        function index_of (
                list        : Pcte_oo.method_request_ids.sequence;
                item        : Pcte_oo.method_request_id;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.integer;
(72)        function are_equal (
                first       : Pcte_oo.method_request_ids.sequence;
                second      : Pcte_oo.method_request_ids.sequence;
                status      : Pcte_error.handle := EXCEPTION_ONLY)
                return      Pcte.boolean;
(73)        procedure normalize (
                list        : in out    Pcte_oo.method_request_ids.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(74)        procedure discard (
                list        : in out    Pcte_oo.method_request_ids.sequence;
                status      : in        Pcte_error.handle := EXCEPTION_ONLY);
(75)    private
(76)        implementation-defined
(77)    end method_request_ids;
```

A.1.2 Object-oriented invocation management operations

```
(1)     package process is
(2)         -- G.2.2.1 PROCESS_ADOPT_CONTEXT
(3)         procedure adopt_context (
                context_adoptions   : in Pcte_oo.context_adoptions.sequence)
(4)     end process;
```

```
(5)   package request is

(6)       -- G.2.2.2 REQUEST_INVOKE

(7)       function invoke (
          request              : Pcte_oo.method_request;
          context_adoptions    : Pcte_oo.context_adoptions)
          return                Pcte_oo.method_request_id;

(8)       -- G.2.2.3 REQUEST_SEND

(9)       function send (
              request              : Pcte_oo.method_request;
              context_adoptions    : Pcte_oo.context_adoptions.sequence)
              return                Pcte_oo.method_request_id;

(10)      -- G.2.2.4 REQUEST_SEND_MULTIPLE

(11)      function send_multiple (
              requests             : Pcte_oo.method_requests.sequence;
              context_adoptions    : Pcte_oo.context_adoptions.sequence)
              return                Pcte_oo.method_request_ids.sequence;

(12)  end request;
```

A.2 Object-oriented schema management

```
(1)   package sds is
```

A.2.1 Object-oriented schema management datatypes

```
(1)       type interface_scope is (NO_OPERATION, ALL_OPERATIONS);
```

A.2.2 Object-oriented schema management operations

```
          -- G.3.2.1 SDS_APPLY_INTERFACE_TYPE

(1)       procedure apply_interface_type (
              sds              : in   Pcte.object_reference;
              interface_type   : in   Pcte.type_reference;
              type             : in   Pcte.type_reference);

          -- G.3.2.2 SDS_APPLY_OPERATION_TYPE

(2)       procedure apply_operation_type (
              sds              : in   Pcte.object_reference;
              operation_type   : in   Pcte.type_reference;
              type             : in   Pcte.type_reference);
```

```
-- G.3.2.3 SDS_CREATE_DATA_PARAMETER_TYPE

(3)  function create_data_parameter_type (
         sds            : Pcte.object_reference;
         local_name     : Pcte.name := "";
         data_type      : Pcte.type_reference)
         return           Pcte.type_reference;

-- G.3.2.4 SDS_CREATE_INTERFACE_PARAMETER_TYPE

(4)  function create_interface_parameter_type (
         sds                : Pcte.object_reference;
         local_name         : Pcte.name := "";
         interface_type     : Pcte.type_reference)
         return               Pcte.type_reference;

-- G.3.2.5 SDS_CREATE_INTERFACE_TYPE

(5)  function create_interface_type (
         sds                : Pcte.object_reference;
         local_name         : Pcte.name := "";
         parents            : Pcte.type_references.sequence;
         new_operations     : Pcte.type_references.sequence)
         return               Pcte.type_reference;

--   G.3.2.6 SDS_CREATE_OBJECT_PARAMETER_TYPE

(6)  function create_data_parameter_type (
         sds            : Pcte.object_reference;
         local_name     : Pcte.name := "";
         object_type    : Pcte.type_reference)
         return           Pcte.type_reference;

--   G.3.2.7 SDS_CREATE_OPERATION_TYPE

(7)  function create_operation_type (
         sds            : Pcte.object_reference;
         local_name     : Pcte.name := "";
         parameters     : Pcte.type_references.sequence;
         return_value   : Pcte.type_reference)
         return           Pcte.type_reference;

--   G.3.2.8 SDS_IMPORT_INTERFACE_TYPE

(8)  procedure import_interface_type (
         to_sds         : in    Pcte.object_reference;
         from_sds       : in    Pcte.object_reference;
         type           : in    Pcte.type_reference;
         local_name     : in    Pcte.name := ""
         import_scope : in    Pcte_oo.sds.interface_scope);
```

```
            -- G.3.2.9 SDS_IMPORT_OPERATION_TYPE
(9)         procedure import_operation_type (
                to_sds      : in    Pcte.object_reference;
                from_sds    : in    Pcte.object_reference;
                type        : in    Pcte.type_reference;
                local_name  : in    Pcte.name := "");

            -- G.3.2.10 SDS_UNAPPLY_INTERFACE_TYPE
(10)        procedure unapply_interface_type (
                sds             : in    Pcte.object_reference;
                interface_type  : in    Pcte.type_reference;
                type            : in    Pcte.type_reference);

            -- G.3.2.11 SDS_UNAPPLY_OPERATION_TYPE
(11)        procedure unapply_operation_type (
                sds             : in    Pcte.object_reference;
                operation_type  : in    Pcte.type_reference;
                type            : in    Pcte.type_reference);
(12)    end sds;
(13)end Pcte_oo;
```

Index of abstract operations

Index of Ada subprograms

Index of Ada datatypes

Printed copies can be ordered from:

ECMA
114 Rue du Rhône
CH-1204 Geneva
Switzerland

Fax: +41 22 849.60.01
Internet: documents@ecma.ch

Files can be downloaded from our FTP site, **ftp.ecma.ch,** logging in as **anonymous** and giving your E-mail address as **password**. This Standard is available from library **ECMA-ST** as a compacted, self-expanding file in MSWord 6.0 format (file E162-DOC.EXE) and as an Acrobat PDF file (file E162-PDF.PDF). File E162-EXP.TXT gives a short presentation of the Standard.

Our web site, http://www.ecma.ch, gives full information on ECMA, ECMA activities, ECMA Standards and Technical Reports.

ECMA

114 Rue du Rhône
CH-1204 Geneva
Switzerland

This Standard ECMA-162 is available free of charge in printed form and as a file.

See inside cover page for instructions