Standard ECMA-158
4th Edition - December 1997

ECMA

Standardizing Information and Communication Systems

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

Standard ECMA-158
4th Edition - December 1997

ECMA

Standardizing Information and Communication Systems

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

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 C Language Binding for ECMA PCTE was started in early 1990, as the Abstract Specification became stable. The C Language Binding was the first binding of ECMA PCTE to be developed, though the strategy for it was developed in parallel with that for the Ada 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 June 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 2 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 13719-2, 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-228 (Extensions for Support of Fine-Grain Objects) and ECMA-256 (Object Orientation Extensions). It is aligned with the second edition of ISO/IEC 13719-2.

Adopted as 4th Edition of Standard ECMA-158 by the General Assembly of December 1997.

Contents

1 Scope

(1) This ECMA Standard defines the binding of the Portable Common Tool Environment (PCTE), as specified in ECMA-149, to the C 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, Implementation Limits.

(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 C 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 C 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. Members of IEC and ISO maintain registers of currently valid International Standards.

(2)	ECMA-149	Portable Common Tool Environment (PCTE) - Abstract Specification (4th edition, December 1997)
(3)	ECMA-162	Portable Common Tool Environment (PCTE) - Ada 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 9899	Information technology - Programming languages, their environments and system software interfaces - Programming languages - C (1990)
(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. For the C Language binding for each operation, the function prototype syntax is used as defined in ISO 9899.

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 C 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 23.1.2 and 23.2 of ECMA-149.

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

(6) Clause 25 defines the binding of the error conditions specified in annex C of ECMA-149, and defines binding-defined error conditions for the C 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 C programming language standard

(1) This ECMA Standard is a conforming program according to ISO 9899.

7.2 General principles

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

(2) ISO/IEC TR10182: Guidelines for Language Bindings 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 exactly one operation in this ECMA Standard except possibly when the abstract operation has distinct functionality depending on the values of one or more parameters.

(4) Each operation in this ECMA Standard should have the same number of parameters as does the corresponding operation in ECMA-149.

(5) All operations in this ECMA Standard should return an integer status value. All other values returned by the operation should be passed back to the caller via an output parameter. The return value of the operation should indicate success or failure only.

(6) Operation and parameter names should be the same in this ECMA Standard as they are in ECMA-149, with the exception that identifiers with file scope should begin 'Pcte_' and otherwise consist of lowercase letters and underscores. The PCTE standard guarantees that there are no ambiguities in names prefixed by 'Pcte_'.

(7) All additional names introduced in this ECMA Standard which are visible across the interface (except header names, see 8.5) should begin 'Pcte_' and otherwise consist of lowercase letters, underscores and digits, or begin 'PCTE_' and otherwise consist of uppercase letters, underscores and digits. The PCTE standard guarantees that there are no ambiguities in names prefixed by 'PCTE_' or 'Pcte_'.

(8) Wherever practical, types introduced for passing complex data entities between caller and operation (and vice versa) should be private types defined by this ECMA Standard. The principle should only be ignored for reasons of ease of use and efficiency of implementation.

(9) Each simple datatype in ECMA-149 should be mapped to a corresponding type defined in this ECMA Standard. Each implementation of the binding should then be free to map the binding-defined type to an efficient C Language basic type appropriate for the platform of the implementation, within the constraints specified in this ECMA Standard.

(10) A general policy of memory allocation should be adopted; see 7.5.

7.3 Sets and sequences

(1) Some complex data entities to be passed into or retrieved from an operation are defined as sets or sequences of a base type in ECMA-149. Bounded set types are mapped individually to bit-significant natural numbers; unbounded set and sequence types are mapped to a private type, **Pcte_sequence** with operations for creation, population, retrieval and deletion. These operations allow multiple elements of sets and sequences to be set and read in a single operation, from or to an array object of an appropriate base type. Thus, the data for sets and sequences can be easily manipulated using standard C Language paradigms, while allowing the implementation to choose the best implementation for such sets and sequences.

7.4 Character strings

(1) In ECMA-149, two different types are used to represent sequences of characters. String is a sequence of Octets allowing all 8-bit values and Text is a sequence of Latin-1 graphic characters. Contents, string attributes etc., are of type String; keys, type names etc., are of type Text.

(2) In the C Bindings, String is mapped to **Pcte_string** (see 8.2.11). Text is mapped to the native C language string with a possibly fixed length, i.e. char * with operations depending on NUL ('\0') character termination (see 8.2.6).

7.5 Memory allocation

(1) Communication between caller and operation is effected by the transfer of data into an operation via an input parameter or back from an operation via an output parameter. There are two types of such parameters: public and private.

(2) All instances of a public type are allocated and managed by the caller of the operation. All instances of a private type are allocated and managed by the implementation. However, the extraction of data from a private type is again by data transfer in instances of a public type via an operation on the private type. In these cases also, the caller of the operation is required to allocate and manage the instances of the public type. The caller is further required to allocate sufficient space to contain a handle to the private type, the type of which is always a pointer to an internal data structure of undefined form. Operations on the private type are provided to create and discard these internal data structures.

(3) Data stored in an instance of a private type is owned by the implementation. The implementation is responsible for allocating, managing and deallocating the memory used to store this data. Furthermore, after a handle to an instance of a private type has been returned to the caller of an operation, via an output parameter, the implementation is responsible for maintaining the data stored therein, until the caller explicitly indicates that the data is no longer needed, by invoking the discard operation on that instance.

(4) Data stored in an instance of a public type and passed into an operation via an input parameter, is owned by the caller of the operation. The caller is responsible for allocating, managing and deallocating the memory used to store this data. The caller is further responsible for maintaining the data stored therein for the duration of the operation. If the implementation needs to access the data after the operation has completed, the implementation is responsible for allocating additional memory and storing therein a copy of the data.

(5) All the operations that have sequences as "out" parameters will allocate the sequence and return it as result of operation. The user does not need to allocate the sequence in advance: the user only needs to declare a Pcte_sequence variable and pass the address of that variable.

7.6 References and names

(1) Objects, attributes, links, and types are referred to in this ECMA Standard using object references, attribute references, link references and type references, respectively. References are mapped to private datatypes encapsulating two ways of designating an object, attribute, link, or type: by an external and by an internal reference (see clause 23 of ECMA-149).

(2) Beside these references, in this ECMA Standard also attribute names, link names and type names are used to refer to attributes, links, or types. These names represent external references and they are mapped to the native C language string type.

(3) Therefore two different interfaces are provided in this Binding for clauses 9 to 22:

- one interface using names for attributes, links, and types;
- one interface using references for attributes, links, and types. All operations of this interface begin 'Pcte_h_'. These operations are defined if an operation of the previous interface uses attributes, links, or types. Whenever new datatypes are necessary, they also begin 'Pcte_h_' (see 8.7).

7.7 Operation return values

(1) All the operations are mapped to functions which return a Pcte_error_type value, which indicates success (PCTE_NO_ERROR equivalent to PCTE_OK) or failure (one of the other enumeration values of Pcte_error_type) of the operation. All other information that is passed between the caller and the operation is passed via "out" or "in-out" parameters.

7.8 Error conditions

(1) Error conditions which are defined in ECMA-149 and which can be established and returned by the operations defined in this ECMA Standard are described in clause 25.

(2) All binding-defined errors are defined in 25.1.

7.9 Identifiers

(1) Many of the identifiers in ECMA-149 are longer than 31 characters, but no two identifiers are exactly the same within the first 31 characters. The C Programming Language Standard requires that an internal name (i.e., a macro name or an identifier that does not have external linkage) be unique within the first 31 characters. Thus there is no need for any identifiers to be abbreviated in this Binding.

(2) ISO 9899 requires all identifiers of enumeration values to be distinct. Where ECMA-149 uses the same identifier for values of enumeration types bound to different C Language enumeration types, new names have been invented.

(3) In a few cases an abstract operation has been bound to more than one C Language function, to cater for optional parameters; in these cases also, new names have been invented.

7.10 Implementation limits

(1) ECMA-149 defines a set of limits that must be honoured by all implementations of the Language Bindings. Clause 24 describes the binding-defined identifiers for these limit values and the way in which these limits 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 (*C datatypes*).

(2) PCTE datatype names are printed in normal characters.

(3) LI Datatypes names are printed in italics.

(4) C datatype names are printed in bold except in displayed fragments of C.

(5) The mapping from PCTE datatypes to C datatypes is done in two stages via LI datatypes defined in ISO/IEC 11404.

8.1 Mapping of PCTE datatypes to LI datatypes

(1) As far as possible the names of PCTE datatypes are retained 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 or 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 clause 23 of ECMA-149, and is summarized in table 1.

(2) **Table 1 - Mapping of predefined PCTE datatypes**

PCTE datatype	**LI datatype**
Boolean	*boolean*
Integer	*pcte-integer* = *integer* range (*MIN_INTEGER_ATTRIBUTE* .. *MAX_INTEGER_ATTRIBUTE*)
Natural	*pcte-natural* = *integer* range (*0* .. *MAX_NATURAL_ATTRIBUTE*)
Float	*pcte-float* = *real* (*10, MAX_DIGITS_FLOAT_ATTRIBUTE)* range (*MIN_FLOAT_ATTRIBUTE* .. *MAX_FLOAT_ATTRIBUTE*)
Time	*pcte-time* = *time* (*second*, 10, Pcte_accuracy_factor*)* range (*MIN_TIME_ATTRIBUTE* .. *MAX_TIME_ATTRIBUTE*)
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	*address*
Attribute_reference	*attribute_reference*
Contents_handle	*contents-handle*
Handler	*handler*
Object_reference	*object-reference*
Link_reference	*link-reference*
Position_handle	*position-handle*
Profile_handle	*profile-handle*
Type_reference	*type-reference*

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 *Sequence*; the order of elements being irrelevant.

(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.

(5) PCTE map datatypes are notionally mapped via a new LI datatype generator *Map*; their mappings to C datatypes are defined directly. The mapping of Attribute_assignments is defined in 9.1. The mapping of ACcess_rights, Acl, and Atomic_access_rights is defined in 19.1.

(6) PCTE union datatypes other than enumerations are notionally mapped via the datatype generator *Choice*.

(7) PCTE composite and bracketed datatypes (except private PCTE datatypes, for which see 8.1.2) are mapped to the datatype generator *Record*.

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. In some operations, an index of LI datatype *natural* is used to identify elements in the sequence. The first element is always indexed from 0.

(2) The characterizing operations are: IsEmpty, Head, Tail, Equal, Empty, and Append from *Sequence*, plus Get, Put, Copy, LengthOf, and IndexOf.

(3) Get (s : sequence of *base*, index : Natural) : *base*
is undefined if InOrder (LengthOf(s), index) is true, or is Head(s).if Equal (index, 0) is true; otherwise Get (Tail(s), Add (index, negate(1))).

(4) Put (s : sequence of *base*, e : *base*, index : Natural) :sequence of *base*
is undefined if InOrder (Add (LengthOf(s), 1), index) is true, or is Append (Create(e), s) if Equal (index, 0) is true; otherwise Append (Head(s), Put (Tail(s), e, Add (index, Negate(1)))).

(5) Copy (s : sequence of *base*) : sequence of *base*
is Create() if IsEmpty(s) is true; otherwise Append (Head(s), Copy (Tail(s))).

(6) LengthOf (s : sequence of *base*) : Natural
is 0 if IsEmpty(s); otherwise Add (LengthOf (Tail(s), 1).

(7) IndexOf (s : sequence of *base*, e : *base*) : Natural
is undefined if IsEmpty(s) is true, or is 1 if Equal (Head(s), e) is true; otherwise Add (IndexOf (Tail(s), e), 1).

Bounded-set

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

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

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

8.2 Mapping of LI datatypes to C datatypes

8.2.1 LI datatype boolean

(1) The LI datatype *boolean* is mapped to the C datatype **Pcte_boolean**

(2)
```
typedef <boolean-type> Pcte_boolean;
```
where <boolean-type> is a C integer type.

(3) TRUE is represented by PCTE_TRUE, FALSE is represented by PCTE_FALSE:

```
#define PCTE_TRUE   (Pcte_boolean) 1
#define PCTE_FALSE  (Pcte_boolean) 0
```

(4) In addition, if a value of type **Pcte_boolean** is supplied as or as part of an input parameter, it is taken as TRUE if it is not 0.

(5) **Characterizing operations**

Operation	**C Operation**

Equal(b1, b2) : b	if (b1 == b2)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Not (b1) : b	if (b1 == PCTE_FALSE)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

And (b1, b2) : b	b = b1 && b2;
Or (b1, b2) : b	b = b1 | b2;

8.2.2 LI datatype pcte-integer

(1) The LI datatype *pcte-integer* is mapped to the C datatype **Pcte_integer**.

(2)
```
typedef <integer-type> Pcte_integer;
```
where <integer-type> is a C integer type including the range MIN_INTEGER_ATTRIBUTE to MAX_INTEGER_ATTRIBUTE inclusive.

(3) **Characterizing operations**

Operation	**C Operation**

Equal (i1, i2) : b	if (i1 == i2)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

InOrder (i1, i2) : b	if (i1 <= i2)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

NonNegative (i) : b	if (i >= 0)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Negate (i) : i	i = -i1;
Add (i1, i2) : i	i = i1 + i2;
Multiply (i1, i2) : i	x * y;

8.2.3 LI datatype pcte-natural

(1) The LI datatype *pcte-natural* is mapped to the C datatype **Pcte_natural**.

(2)
```
typedef <natural-type> Pcte_natural;
```

where <natural-type> is a C unsigned integer type including the range 0 to MAX_NATURAL_ATTRIBUTE inclusive.

(3) Furthermore, the unsigned integral datatype in C Language operates under modular arithmetic rules; i.e. the result of any arithmetic operation is always reduced modulo the largest representable value. Therefore, ULONG_MAX + 1 = 0, where ULONG_MAX is the largest value of the chosen base unsigned integer value.

(4) **Characterizing operations**

Operation	**C Operation**

Equal (n1, n2) : b	if (n1 == n2)
	  b = PCTE_TRUE;
	else
	  b = PCTE_FALSE;

InOrder (n1, n2) : b	if (n1 <= n2)
	  b = PCTE_TRUE;
	else
	  b = PCTE_FALSE;

Add (n1, n2) : n	n = n1 + n2;
Multiply (n1, n2) :n	n = n1 * n2;

8.2.4 LI datatype pcte-float

(1) The LI datatype *pcte-float* is mapped to the C datatype **Pcte_float**.

(2)
```
typedef <float-type> Pcte_float;
```

where <float-type> is a C floating type including the range MIN_FLOAT_ATTRIBUTE to MAX_FLOAT_ATTRIBUTE inclusive, with an accuracy of at least MAX_DIGITS_FLOAT_ATTRIBUTE decimal digits, and able to represent SMALLEST_FLOAT_ATTRIBUTE.

(3) **Characterizing operations**

Operation	**C Operation**

Equal (f1, f2) : b	if (f1 == f2)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

InOrder (f1, f2) : b	if (f1 <= f2)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

NonNegative (f) : b	if (f >= 0.0)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Negate (f1) : f	f = -f1;
Add (f1, f2) : f	f = f1 + f2;
Multiply (f1, f2) : f	f = f1 * f2;
Reciprocal (f1) : f	f = 1.0 / f1;

8.2.5 LI datatype pcte-time

(1) The LI datatype *pcte-time* is mapped to the C datatype **Pcte_time**, where Pcte_time_accuracy_factor is an implementation-defined constant for the resolution of **'Pcte_time'**. Pcte_reference_time is the default initial value for time attributes.

(2) typedef time_t Pcte_time;

(3) #define Pcte_time_accuracy_factor (Pcte_natural) <implementation-defined>

(4) #define Pcte_reference_time (Pcte_time) <implementation-defined>

(5) The datatype **time_t** is an arithmetic datatype that holds values representing time. The encoding of the calendar time within a value of type **time_t** is undefined. Functions on values of type **time_t** are provided to convert such a value into a meaningful and usable representation of calendar time.

(6) The implementation shall provide a constant defining an optional time parameter in the operation Pcte_object_set_time_attributes.

(7) # define Pcte_null_time (Pcte_time) <implementation-defined>

(8) **Characterizing operations**

Operation	**C Operation**

Equal (t1, t2) : b	if (difftime (t1, t2) < radix_factor)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

InOrder (t1, t2) : b	if (difftime (t1, t2) >= 0)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Difference (t1, t2) : i	i = (Pcte_integer) difftime(t1,t2);
Extend.res1tores2 (t1) : t2	t2 = t1 * radix_factor;
Round.res1tores2 (t1) : t2	t2 = floor (t1 + radix_factor);

(9) In this table radix_factor is $10^{-Pcte\_time\_accuracy\_factor}$.

8.2.6 LI datatype pcte-text

(1) The LI datatype *pcte-text* is mapped to the native C language null terminated string type with a fixed length. The character repertoire of the characters of such a sequence are the graphic characters of ISO 8859-1. **Pcte_octet** is used to represent this character repertoire. Attribute names, enumeral type images, exact identifier, keys, machine names, node names, link names, names, type names, and type names in SDSs are of type Text.

(2)
```
#define PCTE_MAX_XXX_SIZE <implementation-defined>
```

(3)
```
typedef Pcte_octet Pcte_xxx[PCTE_MAX_XXX_SIZE + 1];
```

(4) XXX is one of the following PCTE datatypes: Attribute_name, Enumeral_type_image, Exact_identifier, Key, Link_name, Machine_name, Name, Node_name, and Type_name, or Type_name_in_sds. All values of PCTE_XXX_SIZE defined in this Binding are minimum values, which have to be respected by an implementation.

(5) **Characterizing operations**

Operation	C Operation

Equal (s1, s2) : b	if (strcomp (s1, s2) == 0)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Empty () : s	s[0] = '\0';
Head (s1) : c	c = s[0];

Tail (s1) : s2	if (s1[0] != '\0') {
	   s2 = malloc (strlen(s1));
	   (void) strncpy (s2,s1+1,strlen(s1));
	}

Append (s1, e) : s2	s2 = malloc (strlen(s1)+2);
	(void) strcpy (s2,s1);
	(void) strcat (s2,"e");

IsEmpty (s) : b	if (s1[0] == '\0')
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

(6) In this table memory errors like access at invalid addresses are not recognized.

8.2.7 LI datatype octet

(1) The LI datatype *octet* is mapped to the C datatype **char**.

(2)
```
typedef char Pcte_octet;
```

(3) **Characterizing operations**

Operation	C Operation

Equal (o1, o2) : b	if (o1 == o2)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

8.2.8 LI enumerated datatype pcte-xxx

(1) The LI datatype *pcte-xxx*, defined as *enumerated* (value1, value2, ...), corresponds to the PCTE enumeration datatype xxx (where the values of xxx are VALUE1, VALUE2, ...). It is mapped to the C datatype **Pcte_xxx**, defined as follows.

(2) Case (1): if a bounded set of the enumeration type is not required the mapping is:

(3)
```
typedef enum {PCTE_VAL1, PCTE_VAL2, ...} Pcte_xxx;
```

(4) Case (2): if a bounded set of the enumeration type is required the mapping is as defined in 8.2.12.

(5) **Characterizing operations**

Operation	**C Operation**

Equal (e1, e2) : b	if (e1 == e2)
	  b = PCTE_TRUE;
	else
	  b = PCTE_FALSE;

InOrder (e1, e2) : b	if (e1 <= e2)
	  b = PCTE_TRUE;
	else
	  b = PCTE_FALSE;

Successor (e1) : e2	e2 = e1 + 1;

8.2.9 LI private datatypes

(1) Each LI private datatype *xxx* is mapped to an opaque C datatype *Pcte_xxx*.

(2)
```
typedef void *Pcte_xxx;
```

8.2.10 LI datatype generator pcte-sequence

(1) The LI datatypes of the family *pcte-sequence* are mapped to the C datatype **Pcte_sequence** (except for *pcte-sequence* of *octet*, see 8.2.7).

(2) Since **Pcte_sequence** is an opaque type in the C Language, objects of this type may be manipulated only via the binding defined operations (see 8.5.1).

(3) **Characterizing operations**

Operation	**C Operation**
Equal (s1, s2) : b	Pcte_sequences_are_equal (s1, s2, &b);
Empty () : s	Pcte_sequence_create (-, NULL, 0, &s);
Head (s) : e	Pcte_sequence_get (s, 0, &e);

Tail (s1) : s2	Pcte_sequence_create (-, NULL, 0, &s2);
	Pcte_sequence_get_length (s1, &n);
	Pcte_sequence_copy (s1, s2, 0, 1, n-1);

Append (s1, e) : s2	Pcte_natural n;
	Pcte_sequence_get_length (s1, &n);
	Pcte_sequence_create (–, NULL, 0, &s2);
	Pcte_sequence_copy (s1, s2, 0, 0, n);
	Pcte_sequence_insert (s2, n, &e);

IsEmpty (s) : b	Pcte_natural n;
	Pcte_sequence_get_length (s, &n);
	if (n == 0)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Copy (s1) : s2	Pcte_natural n;
	Pcte_sequence_get_length (s1, &n);
	Pcte_sequence_create (–, NULL, 0, &s2);
	Pcte_sequence_copy (s1, s2, 0, 0, n);

Get (s, n) : e	Pcte_sequence_get (s, n, &e);

Put (s1, e, n) ; s2	Pcte_natural n;
	Pcte_sequence_get_length (s1, &n);
	Pcte_sequence_create (–, NULL, 0, &s2);
	Pcte_sequence_copy (s1, s2, 0, 0, n);
	Pcte_sequence_insert (s2, n, &e);

LengthOf (s) : n	Pcte_sequence_get_length (s, &n);
IndexOf (s, e) : n	Pcte_sequence_get_index (s, &e, &n);

(4) If a sequence is to be created using Pcte_sequence_create, a type of **Pcte_sequence_type** has to be supplied. In the table given above, this is indicated using '-'.

(5) The sequence operations, Pcte_sequence_xxx, which are used above are defined in 8.5.1.

8.2.11 LI datatype pcte-string

(1) The LI datatype *pcte-string*, which is a *pcte-sequence* of *octets*, is mapped to the C datatype **Pcte_string**. Attribute values, contents, control data device characteristics, foreign devices,

foreign names, foreign parameters, messages, pathnames, process data, relative pathnames, security labels, and volume characteristics are Strings.

(2)
```
typedef struct {
    Pcte_natural   size;
    Pcte_octet *array;
} Pcte_string;
```

(3) If a value of type **Pcte_string** is passed as or as part of an input parameter, **size** defines the valid number of octets given by **array**. If **size** is bigger than the number of octets allocated in **array**, the error PCTE_ACCESS_AT_INVALID_ADDRESS is raised.

(4) If a value of type **Pcte_string** is passed as or as part of an output parameter, the space for the string can be either allocated by the caller or by the implementation:

- if the supplied **size** is set to 0, the space is allocated by the implementation; in this case it is in the responsibility of the caller to discard the space when the string is no longer needed, by using the operation Pcte_string_discard.
- if the supplied **size** is non-zero, it indicates the number of octets allocated by the user in array.

The implementation sets **size** to the number of returned octets, provided by the implementation in **array**. If there is not enough space in **array**, the implementation raises the error PCTE_STRING_TOO_SHORT or in case of a wrong set **size** PCTE_ACCESS_AT_INVALID_ADDRESS.

(5) In order to improve usability, there are four exceptions to this rule: CONTENTS_READ, CONTENTS_WRITE, PROCESS_PEEK, and PROCESS_POKE. In all cases an array of **Pcte_octet** and an additional size parameter has to be supplied. For these two parameters the same rules apply for incoming and outgoing parameters as for *pcte-string*. In addition, pathname and relative pathname are mapped to the native C language string type (see 23.1). For these exceptions no characterizing operations are defined.

(6) The following operation is provided.

Pcte_string_discard

(7)
```
Pcte_error_type Pcte_string_discard (
    Pcte_string    string
);
```

(8) Pcte_string_discard discards the space allocated by the implementation for string *string*.

(9) **Characterizing operations**

Operation	**C Operation**

Equal (s1, s2) : b	if ((s1.size == s2.size)
	&& (memcmp(s1.array, s2.array, s1.size) == 0))
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Empty () : s	s.size == 0;
Head (s) : e	c = s.array[0];

Tail (s1) : s2	s2.size = s1.size - 1;
	s2.array = malloc (s2.size);
	memcpy (s2.array, s1.array, s2.size);

Append (s1, e) : s2	s2@s2.size = s1.size + 1;
	s2.array = malloc (s2.size);
	memcpy (s2.array, s1.array, s2.size);
	s2.array [s2.size] = e;

IsEmpty (s) : b	if (s.size == 0)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Copy (s1) : s2	s2.size = s1.size;
	s2.array = malloc(s2.size);
	memcpy(s2.array, s1.array, s2.size);

Get (s, n) : e	e = s.array[n];

Put (s1, e, n) ; s2	s2.size = s1.size;
	s2.array = malloc(s2.size);
	memcpy(s2.array, s1.array, s2.size);
	s2.array[n] = e;

LengthOf (s) : n	n = s.size;

IndexOf (s, e) : n	n = 0;
	while (s.array[n] != e) n++;

(10) Note that in this table memory errors like access at invalid addresses are not recognized.

8.2.12 LI datatype generator bounded-set

(1) Each LI datatype *bounded-set of (base)* is mapped to a different C datatype, defined as **Pcte_natural** (see 8.2.3), with an enumeration definition mapping the names of element values to bit positions. A natural value represents a set including an element value if the corresponding bit is 1. Thus a bounded set <bounded-set>, where <bounded-set> is **Pcte_access_events**, **Pcte_categories**, **Pcte_definition_mode_values**, **Pcte_discretionary_access_modes**, or

Pcte_work_status, with possible element values VALUE1, VALUE2, etc. is mapped as follows, making use of the corresponding enumerated types <enumerated-type>, where <enumerated-type> is **Pcte_access_event**, **Pcte_category**, **Pcte_definition_mode_value**, **Pcte_discretionary_access_mode**, or **Pcte_work_status_item**, respectively:

(2)
```
enum {
    PCTE_VALUE1 =   1<<0,
    PCTE_VALUE2 =   1<<1,
    PCTE_VALUE3 =   1<<2,
     ...
} <enumerated-type>;
```

(3)
```
typedef Pcte_natural <bounded-set>;
```

(4) **Characterizing operations**

Operation	**Ada Operation**

Equal (s1, s2) : b	if (s1 == s2)
	  b = PCTE_TRUE;
	else
	  b = PCTE_FALSE;

Empty () : s	s = 0;

IsIn (s, e) : b	if (s1 & e)
	  b = PCTE_TRUE;
	else
	  b = PCTE_FALSE;

Subset (s1, s2) : b	if (s1 == (s2 & s1))
	  b = PCTE_TRUE;
	else
	  b = PCTE_FALSE;

Complement (s1) : s2	s2 = ~s1;
Union (s1, s2) : s3	s3 = s1 | s2;
Intersection (s1, s2) : s3	s3 = s1 & s2;
SetOf (e) : s	s = e;

Select (s) : e	for (e=1; ; e<<1)
	  if (s & e) break;

8.2.13 LI datatype generator choice

(1) The LI datatype *pcte-xxx = choice (alt-1, alt-2, ...)*\fP is mapped to a C union datatype as follows:

(2) Each value is discriminated by an enumeration, the discriminator, with its origin type. The position of the enumeration value of the discriminator corresponds to the position of the type of the choice, i.e. if the discriminator has the value PCTE_TYPE_1, the value of the choice is of type Pcte_type_1, etc.

(3)

```
typedef struct {
    enum {
        PCTE_TYPE_1,
        PCTE_TYPE_2,
    ...
    } union_type;
    union {
        Pcte_type_1   alt-1;
        Pcte_type_2   alt-2;
        ...;
    } choice;
} Pcte_xxx ;
```

(4) **Characterizing operations**

Operation	**C Operation**

Equal (c1, c2) : b	if (c1 == c2)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Tag.type-n (x, type-n) : c	c.union_type = PCTE_TYPE_N
	c.choice = x;

IsType.type-n (c) : b	if (c.union_type == PCTE_TYPE_N)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

Case.type-n (c) : x	x = c.choice;

8.2.14 LI datatype record

(1) The LI datatype generator *Record* is mapped to the C language struct type as follows:

(2)

```
typedef struct {
    Pcte_type_1   component_1;
    Pcte_type_2   component_2;
    Pcte_type_3   component_3;
    ...;
} Pcte_xxx ;
```

(3) **Characterizing operations**

Operation	**C Operation**

Equal (r1, r2) : b	if (r1 == r2)
	   b = PCTE_TRUE;
	else
	   b = PCTE_FALSE;

FieldSelect.c (r) : c	c = r.c;

FieldReplace (r1, c)) : r2	r2 = r1;
	r2.c = c;

8.3 Private datatypes

(1) Private Datatypes are datatypes with a hidden representation, i.e. the implementation-defined representation of such a datatype is not defined in any header. These datatypes are denoted in the C language using **void ***. Table 3 shows all private types which are defined in this binding with the corresponding operations to create and discards instances of these types.

Table 3 - Creation and deletion of private datatypes

Private C datatype	**Create operations**	**Discard operations**
Pcte_attribute_ reference	Pcte_type_reference_copy Pcte_type_reference_set	Pcte_type_reference_unset
Pcte_contents_ handle	Pcte_contents_open Pcte_contents_get_handle_from_key Pcte_contents_handle_duplicate Pcte_contents_handle_duplicate_to_key	Pcte_contents_close
Pcte_link_reference	Pcte_link_reference_copy Pcte_link_reference_set	Pcte_link_reference_unset
Pcte_object_ reference	Pcte_object_reference_copy Pcte_object_reference_set_absolute Pcte_object_reference_set_relative	Pcte_object_reference_unset
Pcte_position_ handle	Pcte_contents_get_position	Pcte_position_handle_discard Pcte_contents_close
Pcte_profile_handle	Pcte_process_profiling_on	Pcte_process_profiling_off
Pcte_sequence	Pcte_sequence_create	Pcte_sequence_discard
Pcte_type_reference	Pcte_type_reference_copy Pcte_type_reference_set	Pcte_type_reference_unset

8.4 References and names

(1) The LI datatypes *attribute_reference*, *link_reference* and *type_reference* are mapped to the C datatypes **Pcte_attribute_name**, **Pcte_link_name**, and **Pcte_type_name**.

(2) If an attribute reference, link reference, or type reference is passed as part of a parameter, the corresponding parts of the LI datatype of that parameter is also mapped to the C datatypes **Pcte_attribute_name**, **Pcte_link_name**, or **Pcte_type_name**.

(3) Whenever a parameter of an operation in clause 9 to 22 is either an attribute reference, link reference or type reference, a second interface is defined beginning 'Pcte_h_' instead of 'Pcte_'. In this case, the LI datatypes *attribute_reference*, *link_reference* and *type_reference* are mapped to the C datatypes **Pcte_attribute_reference**, **Pcte_link_reference**, and **Pcte_type_reference** (see clause 23).

(4) It may be necessary that if an attribute reference, link reference, or type reference is passed as part of a parameter of the C datatype **Pcte_xxx**, a new C datatype **Pcte_h_xxx** is defined using the C datatypes **Pcte_attribute_reference**, **Pcte_link_reference**, and **Pcte_type_reference**.

(5) All definitions concerning operations beginning with 'Pcte_' also hold for 'Pcte_h_'.

8.5 C private type Pcte_sequence

(1) The new LI datatype *Sequence* is mapped to a family of C datatypes defined by the C datatype **Pcte_sequence**, defined as follows:

(2) typedef void *Pcte_sequence;

(3) #define Pcte_null_sequence (Pcte_sequence) NULL

(4) In addition, there is for each LI datatype *xxx* which is a *sequence*, except *text* and *Pcte_string*, a C datatype with the corresponding name **Pcte_xxx**. Instead of **Pcte_h_attribute_references**, **Pcte_h_link_references**, and **Pcte_h_type_references**, the C datatypes **Pcte_attribute_names**, **Pcte_link_names**, and **Pcte_type_names** are used.

(5) As ACL and attribute assignments are also defined as sequences (see 9.1, 19.1) corresponding C datatypes for these PCTE datatypes are also provided.

(6) typedef Pcte_sequence Pcte_accounting_log;

(7) typedef Pcte_sequence Pcte_acl;

(8) typedef Pcte_sequence Pcte_audit_file;

(9) typedef Pcte_sequence Pcte_attribute_assignments;

(10) typedef Pcte_sequence Pcte_h_attribute_assignments;

(11) typedef Pcte_sequence Pcte_attribute_names;

(12) typedef Pcte_sequence Pcte_attribute_references;

(13) typedef Pcte_sequence Pcte_buffer;

(14) typedef Pcte_sequence Pcte_confidentiality_criteria;

(15) typedef Pcte_sequence Pcte_enumeration_value_type;

(16) typedef Pcte_sequence Pcte_h_enumeration_value_type;

(17) typedef Pcte_sequence Pcte_enumeration_value_type_in_sds;

(18) typedef Pcte_sequence Pcte_general_criteria;

(19) typedef Pcte_sequence Pcte_integrity_criteria;

(20) typedef Pcte_sequence Pcte_key_types;

(21) typedef Pcte_sequence Pcte_h_key_types;

(22) typedef Pcte_sequence Pcte_key_types_in_sds;

(23) typedef Pcte_sequence Pcte_link_set_descriptors;

(24) typedef Pcte_sequence Pcte_h_link_set_descriptors;

(25) typedef Pcte_sequence Pcte_link_names;

(26) typedef Pcte_sequence Pcte_link_references;

(27) typedef Pcte_sequence Pcte_message_types;

(28) typedef Pcte_sequence Pcte_name_sequence;

(29) typedef Pcte_sequence Pcte_object_criteria;

(30) typedef Pcte_sequence Pcte_object_references;

(31) typedef Pcte_sequence Pcte_type_names;

(32) typedef Pcte_sequence Pcte_type_names_in_sds;

(33) typedef Pcte_sequence Pcte_type_references;

(34)
```
typedef Pcte_sequence Pcte_user_criteria;
```
(35)
```
typedef Pcte_sequence Pcte_volume_infos;
```
(36)
```
typedef Pcte_sequence Pcte_parameters_items;
```
(37)
```
typedef Pcte_sequence Pcte_method_requests;
```
(38)
```
typedef Pcte_sequence Pcte_context_adoptions;
```
(39)
```
typedef Pcte_sequence Pcte_method_request_ids;
```

(40) The following additional datatypes are used in the operational interfaces to these private types:

(41)
```
typedef enum {
    PCTE_ACCOUNTING_FILE, PCTE_ACL,
    PCTE_AUDIT_FILE, PCTE_ATTRIBUTE_ASSIGNMENTS,
    PCTE_H_ATTRIBUTE_ASSIGNMENTS,
    PCTE_ATTRIBUTE_NAMES, PCTE_ATTRIBUTE_REFERENCES,
    PCTE_BUFFER, PCTE_CONFIDENTIALITY_CRITERIA,
    PCTE_ENUMERATION_VALUE_TYPE,
    PCTE_H_ENUMERATION_VALUE_TYPE,
    PCTE_ENUMERATION_VALUE_TYPE_IN_SDS,
    PCTE_GENERAL_CRITERIA, PCTE_INTEGRITY_CRITERIA,
    PCTE_KEY_TYPES, PCTE_H_KEY_TYPES,
    PCTE_KEY_TYPES_IN_SDS, PCTE_LINK_NAMES,
    PCTE_LINK_SET_DESCRIPTORS, PCTE_H_LINK_SET_DESCRIPTORS,
    PCTE_LINK_REFERENCES, PCTE_MESSAGE_TYPES,
    PCTE_NAME_SEQUENCE, PCTE_OBJECT_CRITERIA,
    PCTE_OBJECT_REFERENCES, PCTE_TYPE_NAMES,
    PCTE_TYPE_NAMES_IN_SDS, PCTE_TYPE_REFERENCES,
    PCTE_USER_CRITERIA, PCTE_VOLUME_INFOS,
    PCTE_PARAMETER_ITEMS, PCTE_METHOD_REQUESTS,
    PCTE_CONTEXT_ADOPTIONS, PCTE_METHOD_REQUEST_IDS }
Pcte_sequence_type;
```

(42)
```
typedef void *Pcte_sequence_element;
```
(43)
```
typedef void *Pcte_array_of_sequence_elements;
```

(44) Values of the type **Pcte_sequence_element** are used to set or retrieve a single element of a sequence using the operations Pcte_put_element or Pcte_get_element. In both cases, a pointer to a single value of the sequence element type has to be provided. For example, if the sequence is of type **Pcte_object_references**, a pointer to a value of type **Pcte_object_reference** has to be provided.

(45) **Pcte_array_of_sequence_elements** is used to set or retrieve several elements to or from a sequence using the operations Pcte_sequence_create, Pcte_sequence_insert_elements, and Pcte_sequence_get_elements. In this case, a pointer to an array of the sequence element type has to be provided. For example, if the sequence is of type **Pcte_object_references**, a pointer to an array of **Pcte_object_reference** has to be provided.

(46) The enumeration type **Pcte_sequence_type** is used to indicate the C element type of the sequence as follows:

Table 4 - Sequence types

enumeration value	**C element type**
PCTE_ACCOUNTING_FILE	Pcte_accounting_record
PCTE_ACL	Pcte_acl_entry
PCTE_AUDIT_FILE	Pcte_auditing_record
PCTE_ATTRIBUTE_ASSIGNMENTS	Pcte_attribute_assignment
PCTE_H_ATTRIBUTE_ASSIGNMENTS	Pcte_h_attribute_assignment
PCTE_ATTRIBUTE_NAMES	Pcte_attribute_name
PCTE_ATTRIBUTE_REFERENCES	Pcte_attribute_reference
PCTE_BUFFER	Pcte_natural
PCTE_CONFIDENTIALITY_CRITERIA	Pcte_confidentiality_criterion
PCTE_ENUMERATION_VALUE_TYPE	Pcte_type_name
PCTE_H_ENUMERATION_VALUE_TYPE	Pcte_type_reference
PCTE_ENUMERATION_VALUE_TYPE_IN_SDS	Pcte_type_name_in_sds
PCTE_GENERAL_CRITERIA	Pcte_general_criterion
PCTE_INTEGRITY_CRITERIA	Pcte_integrity_criterion
PCTE_KEY_TYPES	Pcte_type_name
PCTE_H_KEY_TYPES	Pcte_type_reference
PCTE_KEY_TYPES_IN_SDS	Pcte_type_name_in_sds
PCTE_LINK_NAMES	Pcte_link_name
PCTE_LINK_SET_DESCRIPTORS	Pcte_link_set_descriptor
PCTE_H_LINK_SET_DESCRIPTORS	Pcte_h_link_set_descriptor
PCTE_LINK_REFERENCES	Pcte_link_reference
PCTE_MESSAGE_TYPES	Pcte_message_type
PCTE_NAME_SEQUENCE	Pcte_name
PCTE_OBJECT_CRITERIA	Pcte_object_criterion
PCTE_OBJECT_REFERENCES	Pcte_object_reference
PCTE_TYPE_NAMES	Pcte_type_name
PCTE_TYPE_NAMES_IN_SDS	Pcte_type_name_in_sds
PCTE_USER_CRITERIA	Pcte_user_criterion

PCTE_VOLUME_INFOS	Pcte_volume_info
PCTE_PARAMETER_ITEMS	Pcte_parameter_item
PCTE_METHOD_REQUESTS	Pcte_method_request
PCTE_CONTEXT_ADOPTIONS	Pcte_context_adoption
PCTE_METHOD_REQUEST_IDS	Pcte_method_request_id

8.5.1 Operations on sequences

(1) The following operations are defined on objects of type **Pcte_sequence**. Each operation returns a status via its return value, which indicates its success or failure. For an interpretation of returned status values, see clause 25, Error Conditions.

(2) In some operations, elements of a sequence can be referenced using an index according to the positions of elements within the sequence. The first element of a sequence has the position 0.

Pcte_sequence_create

(3)
```
Pcte_error_type Pcte_sequence_create (
   Pcte_sequence_type                type,
   Pcte_array_of_sequence_elements   data,
   Pcte_natural                      count,
   Pcte_sequence                     *sequence
);
```

(4) Pcte_sequence_create creates an instance of the sequence type specified by *type*, with *count* element values specified by *data*, and returns a pointer to it in *sequence*.

Pcte_sequence_discard

(5)
```
Pcte_error_type Pcte_sequence_discard (
   Pcte_sequence    *sequence
);
```

(6) Pcte_sequence_discard releases the contents of the sequence pointed to by *sequence* and discards any allocated data structures associated with it. The data structures allocated with each element are also discarded, even if they are themselves sequences. A null pointer is returned in *sequence*. When a sequence of types containing an object reference (i.e. object_references, link_set_descriptors and object criterion) is discarded, the object references contained in the sequence are unset as defined by the operation object_reference_unset.

Pcte_sequence_copy

(7)
```
Pcte_error_type Pcte_sequence_copy (
   Pcte_sequence    source_list,
   Pcte_sequence    destination_list,
   Pcte_natural     index,
   Pcte_natural     source_index,
   Pcte_natural     count
);
```

(8) Pcte_sequence_copy copies *count* subsequent elements of data from the sequence *source_list* from the position *source_index* to the sequence *destination_list* before the element at position *index*. If *index* is greater than the number of elements of *destination_list*, these elements are appended at the end of *destination_list*. Copying sequences is a deep copy operation, and object references are copied as they are with the status and evaluation point. Deallocating the source sequence or any of its elements will not affect the destination.

Pcte_sequence_insert_elements

(9)
```
Pcte_error_type Pcte_sequence_insert_elements (
   Pcte_sequence                     sequence,
   Pcte_natural                      index,
   Pcte_array_of_sequence_elements   data,
   Pcte_natural                      count
);
```

(10) Pcte_sequence_insert_elements inserts the *count* elements specified by *data* into the sequence *sequence* before the element at position *index*.

Pcte_sequence_delete

(11)
```
Pcte_error_type Pcte_sequence_delete (
   Pcte_sequence   sequence,
   Pcte_natural    index,
   Pcte_natural    count
);
```

(12) Pcte_sequence_delete removes the *count* subsequent elements from the sequence *sequence* starting with the element at position *index*.

Pcte_sequences_are_equal

(13)
```
Pcte_error_type Pcte_sequences_are_equal (
   Pcte_sequence   first_sequence,
   Pcte_sequence   second_sequence,
   Pcte_boolean    *equality
);
```

(14) Pcte_sequences_are_equal returns in *equality* the value PCTE_TRUE if the two sequences *first_sequence* and *second_sequence* are equal, and the value PCTE_FALSE otherwise. Two sequences are equal if they contain the same number of elements and the values of corresponding elements are equal.

Pcte_sequence_get_index

(15)
```
Pcte_error_type Pcte_sequence_get_index (
   Pcte_sequence           sequence,
   Pcte_sequence_element   element,
   Pcte_integer            *index
);
```

(16) Pcte_sequence_get_index returns in *index* the index of the first occurrence of the given element *element* in the sequence *sequence* if the element is a member of the sequence; otherwise -1 is returned.

Pcte_sequence_get_length

(17)
```
Pcte_error_type Pcte_sequence_get_length (
    Pcte_sequence     sequence,
    Pcte_natural      *length
);
```

(18) Pcte_sequence_get_length returns in *length* the number of elements of the sequence *sequence*.

Pcte_sequence_get_elements

(19)
```
Pcte_error_type Pcte_sequence_get_elements (
    Pcte_sequence                     sequence,
    Pcte_natural                      index,
    Pcte_array_of_sequence_elements   data,
    Pcte_natural                      count
);
```

(20) Pcte_sequence_get_elements returns *count* elements from the sequence *sequence* into *data*, starting from the element at position *index*.

Pcte_sequence_get

(21)
```
Pcte_error_type Pcte_sequence_get (
    Pcte_sequence           sequence,
    Pcte_natural            index,
    Pcte_sequence_element   element
);
```

(22) Pcte_sequence_get returns in *element* a copy of the element specified by *index* from the sequence *sequence*.

Pcte_sequence_insert

(23)
```
Pcte_error_type Pcte_sequence_insert (
    Pcte_sequence           sequence,
    Pcte_natural            index,
    Pcte_sequence_element   element
);
```

(24) Pcte_sequence_insert inserts in the sequence *sequence* the element *element* immediately before the element specified by *index*. If *index* is not less than the length of *sequence*, the element is appended to *sequence*.

Pcte_sequence_replace

(25)
```
Pcte_error_type Pcte_sequence_replace (
    Pcte_sequence           sequence,
    Pcte_natural            index,
    Pcte_sequence_element   element
);
```

(26) Pcte_sequence_replace replaces in the sequence *sequence* the element specified by *index* with the element *element*. If *index* is not less than the length of *sequence*, the element is appended to *sequence*.

Pcte_sequence_append

(27)
```
Pcte_error_type Pcte_sequence_append (
   Pcte_sequence            sequence,
   Pcte_sequence_element    element
);
```

(28) Pcte_sequence_append appends to the sequence *sequence* the element specified by *element*.

Pcte_sequence_normalize

(29)
```
Pcte_error_type Pcte_sequence_normalize (
   Pcte_sequence    sequence
);
```

(30) Pcte_sequence_normalize reorganizes the sequence *sequence* so that each element value occurs once only, and elements are in an implementation-defined order. Two sequences containing the same element values (i.e. representing the same unbounded set) are equal after normalization.

8.5.2 Error conditions for sequence operators

(1) The following error conditions may be raised by operations on sequences.

- PCTE_SEQUENCE_INVALID_TYPE. Raised by any operation, if the type of a specified sequence does not match the required sequence type for that sequence.
- PCTE_SEQUENCE_BAD_HANDLE. Raised by any operation, if a specified sequence handle is not a valid sequence handle.
- PCTE_SEQUENCE_OUT_OF_DATA. Raised by Pcte_sequence_get_elements, if *sequence* does not contain *count* elements from position *index*. Similarly for Pcte_sequence_copy.
- PCTE_SEQUENCE_INVALID_INDEX. Raised by Pcte_sequence_get, if *sequence* does not contain an element at position *index*.

(2) For further information on error conditions leading to an operational failure, see clause 25, Error Conditions.

8.6 Deriving C function semantics from the abstract specification

(1) Each C function corresponds to the abstract operation in ECMA-149. The semantics of the C function are generally the same as those of the corresponding abstract operation and are derived as follows:

- C function parameters correspond to abstract operation parameters with the same names.
- C function parameter datatypes correspond to the abstract specification datatypes of the corresponding abstract parameters as defined in clause 8.
- Returned values corresponding to results of abstract operations are returned via parametric pointers to objects which contain the returned values. If the result is a mandatory single value of a simple type, the pointer parameter has the same name as the result. In more complicated cases an explanation is given.
- If the abstract operation contains optional parameters, an explanation of the mapping is given.
- All the operations are mapped to functions which return a Pcte_error_type value, which indicates success (PCTE_NO_ERROR equivalent to PCTE_OK) or failure (one of the other

enumeration values of Pcte_error_type) of the operation. The error code PCTE_NO_ERROR is equivalent to PCTE_OK (as both have the value zero). Additionally the global variable Pcte_error_number is set to the same return value of the function after each PCTE call, but it should be noted that in case of multi-threading or queue handlers this global variable cannot be relied upon.

(2) All exceptions to these general rules are defined in the appropriate clause.

(3) Handlers execute as separate threads, subject to implementation-defined restrictions. Apart from that case, neither the C language, nor the PCTE operations provide mechanisms for a program to execute more than one thread; but an implementation may provide such mechanisms, which may be subject to implementation-defined rules.

8.7 Headers

(1) Each implementation of this ECMA Standard must provide a header <Pcte/sequences.h> that contains the definitions of the constant values, datatypes, and operation interface bindings for the manipulation of sequences as defined in 8.3.

(2) Each implementation of this ECMA Standard must provide a set of clause-specific headers that contain the definitions of the constant values, data structures, and operation interface bindings for each clause 9 to 22 of ECMA-149.

(3) Each implementation of this ECMA Standard must provide a header <Pcte/references.h> that contains the definitions of the constant values, datatypes, and operation interface bindings for clause 23 of ECMA-149.

(4) Each implementation of this ECMA Standard must provide a header <Pcte/limits.h> that contains the definitions of the constant values defined in clause 24.

(5) Each implementation of this ECMA Standard must provide a header <Pcte/errors.h> that contains the definitions of the constant values and datatypes for errors defined in annex C of ECMA-149.

(6) In addition, each implementation must provide a single header <Pcte/pcte.h> which is used to include all the other headers.

(7) Each clause-specific header must provide all definitions necessary for the correct functioning of the operations of that clause. This may be achieved by including other headers.

(8) The names of the headers are meant to be interpreted by the C processor as a directory 'Pcte' and files 'xxx.h'. If this does not map appropriately to a file-naming convention then the implementation may change the #include directives appropriately.

8.7.1 The global PCTE header

```
(1)     /* The header <Pcte/pcte.h> */

(2)     #ifndef PCTE_INCLUDED
        #define PCTE_INCLUDED 1

(3)     #include <Pcte/types.h>           /*  clause 8.7.2      */
        #include <Pcte/sequences.h>       /*  clause 8.7.3      */
        #include <Pcte/references.h>      /*  clause 23         */
        #include <Pcte/limits.h>          /*  clause 24         */
        #include <Pcte/errors.h>          /*  clause 25         */
```

(4)
```
#include <Pcte/oms.h>              /*  clause 9      */
#include <Pcte/sms.h>              /*  clause 10     */
#include <Pcte/devices.h>          /*  clause 11     */
#include <Pcte/contents.h>         /*  clause 12     */
#include <Pcte/execution.h>        /*  clause 13     */
#include <Pcte/messages.h>         /*  clause 14     */
#include <Pcte/notification.h>     /*  clause 15     */
#include <Pcte/activities.h>       /*  clause 16     */
#include <Pcte/replication.h>      /*  clause 17     */
#include <Pcte/network.h>          /*  clause 18     */
#include <Pcte/discretionary.h>    /*  clause 19     */
#include <Pcte/mandatory.h>        /*  clause 20     */
#include <Pcte/auditing.h>         /*  clause 21     */
#include <Pcte/accounting.h>       /*  clause 22     */
```

(5)
```
/* header used for cluster management*/
```

(6)
```
#include <Pcte/clusters.h>
```

(7)
```
/* headers used for object orientation*/
```

(8)
```
#include <Pcte/interfaces.h>
#include <Pcte/methods.h>
```

(9)
```
#endif /* !PCTE_INCLUDED */
```

8.7.2 The PCTE basic type header

(1)
```
/* The header <Pcte/types.h> */
```

(2)
```
#ifndef PCTE_TYPES_INCLUDED
#define PCTE_TYPES_INCLUDED 1
```

(3)
```
#include <time.h>
#include <Pcte/errors.h>
```

(4)
```
#define PCTE_OK       0
#define PCTE_ERROR     1
```

(5)
```
typedef <integer-type> Pcte_boolean;
```

(6)
```
#define PCTE_TRUE (Pcte_boolean)0
#define PCTE_FALSE (Pcte_boolean)    1
```

(7)
```
typedef <integer-type> Pcte_integer;
```

(8)
```
typedef <natural-type> Pcte_natural;
```

(9)
```
typedef <float-type> Pcte_float;
```

(10)
```
typedef time_t Pcte_time;
```

(11)
```
#define Pcte_time_accuracy_factor (Pcte_natural) <implementation-defined>
```

(12)
```
#define Pcte_reference_time (Pcte_time) <implementation-defined>
```

(13)
```
#define Pcte_null_time (Pcte_time) <implementation-defined>
```

(14)
```
typedef unsigned char Pcte_octet;
```

(15)
```
typedef struct {
   Pcte_natural   size;
   Pcte_octet *array;
} Pcte_string;
```

(16)
```
/*  Empty string can be used in all operations that have a string as input parameter, where     */
/*  the user wants to pass a null value for the string.                                          */
```

(17)
```
Pcte_string Pcte_null_string = {0, ""};
```

(18)
```
Pcte_error_type Pcte_string_discard (
   Pcte_string    string
);
```

(19)
```
#endif /* !PCTE_TYPES_INCLUDED */
```

8.7.3 The PCTE sequence header

(1) `/* The header <Pcte/sequences.h> */`

(2)
```
#ifndef PCTE_SEQUENCES_INCLUDED
#define PCTE_SEQUENCES_INCLUDED 1
```

(3) `#include <Pcte/types.h>`

(4) `typedef void *Pcte_sequence;`

(5) `#define Pcte_null_sequence (Pcte_sequence) NULL`

(6) `typedef Pcte_sequence Pcte_accounting_log;`

(7) `typedef Pcte_sequence Pcte_audit_file;`

(8) `typedef Pcte_sequence Pcte_attribute_names;`

(9) `typedef Pcte_sequence Pcte_attribute_references;`

(10) `typedef Pcte_sequence Pcte_buffer;`

(11) `typedef Pcte_sequence Pcte_confidentiality_criteria;`

(12) `typedef Pcte_sequence Pcte_enumeration_value_type;`

(13) `typedef Pcte_sequence Pcte_h_enumeration_value_type;`

(14) `typedef Pcte_sequence Pcte_enumeration_value_type_in_sds;`

(15) `typedef Pcte_sequence Pcte_general_criteria;`

(16) `typedef Pcte_sequence Pcte_integrity_criteria;`

(17) `typedef Pcte_sequence Pcte_key_types;`

(18) `typedef Pcte_sequence Pcte_h_key_types;`

(19) `typedef Pcte_sequence Pcte_key_types_in_sds;`

(20) `typedef Pcte_sequence Pcte_link_set_descriptors;`

(21) `typedef Pcte_sequence Pcte_h_link_set_descriptors;`

(22) `typedef Pcte_sequence Pcte_link_names;`

(23) `typedef Pcte_sequence Pcte_link_references;`

(24)
```
typedef Pcte_sequence Pcte_message_types;
```
(25)
```
typedef Pcte_sequence Pcte_name_sequence;
```
(26)
```
typedef Pcte_sequence Pcte_object_criteria;
```
(27)
```
typedef Pcte_sequence Pcte_object_references;
```
(28)
```
typedef Pcte_sequence Pcte_type_names;
```
(29)
```
typedef Pcte_sequence Pcte_type_names_in_sds;
```
(30)
```
typedef Pcte_sequence Pcte_type_references;
```
(31)
```
typedef Pcte_sequence Pcte_user_criteria;
```
(32)
```
typedef Pcte_sequence Pcte_volume_infos;
```
(33)
```
typedef enum {
   PCTE_ACCOUNTING_FILE, PCTE_ACL, PCTE_AUDIT_FILE,
   PCTE_ATTRIBUTE_ASSIGNMENTS, PCTE_H_ATTRIBUTE_ASSIGNMENTS,
   PCTE_ATTRIBUTE_NAMES, PCTE_ATTRIBUTE_REFERENCES, PCTE_BUFFER,
   PCTE_CONFIDENTIALITY_CRITERIA, PCTE_ENUMERATION_VALUE_TYPE,
   PCTE_H_ENUMERATION_VALUE_TYPE,
   PCTE_ENUMERATION_VALUE_TYPE_IN_SDS, PCTE_GENERAL_CRITERIA,
   PCTE_INTEGRITY_CRITERIA, PCTE_KEY_TYPES, PCTE_H_KEY_TYPES,
   PCTE_KEY_TYPES_IN_SDS, PCTE_LINK_NAMES,
   PCTE_LINK_SET_DESCRIPTORS, PCTE_H_LINK_SET_DESCRIPTORS,
   PCTE_LINK_REFERENCES, PCTE_MESSAGE_TYPES, PCTE_NAME_SEQUENCE,
   PCTE_OBJECT_CRITERIA, PCTE_OBJECT_REFERENCES, PCTE_TYPE_NAMES,
   PCTE_TYPE_NAMES_IN_SDS, PCTE_TYPE_REFERENCES,
   PCTE_USER_CRITERIA, PCTE_VOLUME_INFOS
} Pcte_sequence_type;
```
(34)
```
typedef void *Pcte_sequence_element;
```
(35)
```
typedef void *Pcte_array_of_sequence_elements;
```
(36)
```
Pcte_error_type Pcte_sequence_create (
   Pcte_sequence_type                    type,
   Pcte_array_of_sequence_elements       data,
   Pcte_natural                          count,
   Pcte_sequence                         *sequence
);
```
(37)
```
Pcte_error_type Pcte_sequence_discard (
   Pcte_sequence    *sequence
);
```
(38)
```
Pcte_error_type Pcte_sequence_copy (
   Pcte_sequence         source_list,
   Pcte_sequence         *destination_list,
   Pcte_natural          index,
   Pcte_natural          source_index,
   Pcte_natural          count
);
```

(39)
```
Pcte_error_type Pcte_sequence_insert_elements (
    Pcte_sequence                          sequence,
    Pcte_natural                           index,
    Pcte_array_of_sequence_elements        data,
    Pcte_natural                           count
);
```

(40)
```
Pcte_error_type Pcte_sequence_delete (
    Pcte_sequence    sequence,
    Pcte_natural     index,
    Pcte_natural     count
);
```

(41)
```
Pcte_error_type Pcte_sequences_are_equal (
    Pcte_sequence    first_sequence,
    Pcte_sequence    second_sequence,
    Pcte_boolean     *equality
);
```

(42)
```
Pcte_error_type Pcte_sequence_get_index (
    Pcte_sequence              sequence,
    Pcte_sequence_element      element,
    Pcte_integer               *index
);
```

(43)
```
Pcte_error_type Pcte_sequence_get_length (
    Pcte_sequence    sequence,
    Pcte_natural     *length
);
```

(44)
```
Pcte_error_type Pcte_sequence_get_elements (
    Pcte_sequence                          sequence,
    Pcte_natural                           index,
    Pcte_array_of_sequence_elements        data,
    Pcte_natural                           count
);
```

(45)
```
Pcte_error_type Pcte_sequence_get (
    Pcte_sequence              sequence,
    Pcte_natural               index,
    Pcte_sequence_element      element
);
```

(46)
```
Pcte_error_type Pcte_sequence_insert (
    Pcte_sequence              sequence,
    Pcte_natural               index,
    Pcte_sequence_element      element
);
```

(47)
```
Pcte_error_type Pcte_sequence_replace (
    Pcte_sequence              sequence,
    Pcte_natural               index,
    Pcte_sequence_element      element
);
```

(48)
```
Pcte_error_type Pcte_sequence_append (
    Pcte_sequence              sequence,
    Pcte_sequence_element      element
);
```

(49)
```
Pcte_error_type Pcte_sequence_normalize (
    Pcte_sequence    sequence
);
```

(50)
```
#endif /* !PCTE_SEQUENCES_INCLUDED */
```

9 Object management

(1)
```
/* The header <Pcte/oms.h> */
```

(2)
```
#ifndef PCTE_OMS_INCLUDED
#define PCTE_OMS_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
```

9.1 Object management datatypes

(1)
```
typedef enum {
    PCTE_COMPOSITION     =   1<<0,
    PCTE_EXISTENCE       =   1<<1,
    PCTE_REFERENCE       =   1<<2,
    PCTE_DESIGNATION     =   1<<3,
    PCTE_IMPLICIT        =   1<<4
} Pcte_category;
```

(2)
```
typedef Pcte_natural Pcte_categories;
```

(3)
```
#define PCTE_ALL_CATEGORIES (Pcte_natural) (PCTE_COMPOSITION | \
    PCTE_EXISTENCE | PCTE_REFERENCE | PCTE_DESIGNATION | \
    PCTE_IMPLICIT)
```

(4)
```
typedef enum {
    PCTE_BOOLEAN_ATTRIBUTE, PCTE_INTEGER_ATTRIBUTE,
    PCTE_NATURAL_ATTRIBUTE, PCTE_FLOAT_ATTRIBUTE,
    PCTE_STRING_ATTRIBUTE, PCTE_TIME_ATTRIBUTE,
    PCTE_ENUMERATION_ATTRIBUTE
} Pcte_value_type;
```

(5)
```
/*  The PCTE datatype Value_type is a union which also contains the constituent type           */
/*  Enumeration_value_type, which is a sequence of Enumeral_type_nominator.  For               */
/*  convenience, in the operations SDS_GET_ATTRIBUTE_TYPE_PROPERTIES and                      */
/*  WS_GET_ATTRIBUTE_TYPE_PROPERTIES an additional parameter is used to                       */
/*  return an enumeration value type if necessary.  In all other cases it is sufficient to use */
/*  Pcte_value_type.                                                                           */
```

(6)
```
typedef struct {
    Pcte_value_type  type;
    union {
        Pcte_boolean     v_boolean;
        Pcte_integer     v_integer;
        Pcte_natural     v_natural;
        Pcte_float       v_float;
        Pcte_string      v_string;
        Pcte_time        v_time;
        Pcte_natural     v_enumeral_type_position;
    } value;
} Pcte_attribute_value;
```

(7)
```
typedef struct {
    Pcte_attribute_name      attribute;
    Pcte_attribute_value     value;
} Pcte_attribute_assignment;
```

(8)
```
typedef struct {
    Pcte_attribute_reference     attribute;
    Pcte_attribute_value         value;
} Pcte_h_attribute_assignment;
```

(9)
```
#include <Pcte/sequences.h>
```

(10)
```
typedef Pcte_sequence Pcte_attribute_assignments;
```

(11)
```
typedef Pcte_sequence Pcte_h_attribute_assignments;
```

(12)
```
/*  The PCTE datatype attribute assignments, which is a map from attribute reference to a     */
/*  value type, is mapped to the sequence Pcte_attribute_assignments indicated by             */
/*  PCTE_ATTRIBUTE_ASSIGNMENT with the C element datatype                                      */
/*  Pcte_attribute_assignment.  The component attribute represents the domain of the map      */
/*  whereas the component value_type indicates the value of the map.  If a parameter of type  */
/*  Pcte_attribute_assignments is passed as an input parameter to an operation and if there   */
/*  is more than one entry in the sequence with the same value of the component attribute,    */
/*  only the first of these entries is recognized in the operation.                           */
```

(13)
```
typedef enum {
    PCTE_INTERNAL_LINKS, PCTE_EXTERNAL_LINKS, PCTE_ALL_LINKS
} Pcte_link_scope;
```

(14)
```
typedef enum {
    PCTE_EQUAL_TYPE, PCTE_ANCESTOR_TYPE, PCTE_DESCENDANT_TYPE,
    PCTE_UNRELATED_TYPE
} Pcte_type_ancestry;
```

(15)
```
typedef enum {
    PCTE_ANCESTOR_VSN, PCTE_DESCENDANT_VSN, PCTE_SAME_VSN,
    PCTE_RELATED_VSN, PCTE_UNRELATED_VSN
} Pcte_version_relation;
```

(16)
```
typedef enum {
    PCTE_ATOMIC, PCTE_COMPOSITE
} Pcte_object_scope;
```

(17)
```
typedef enum {
    PCTE_ACCESSIBLE, PCTE_INACCESSIBLE,PCTE_UNKNOWN
} Pcte_volume_accessibility;
```

(18)
```
#include <Pcte/devices.h>
```

(19)
```
typedef struct {
    Pcte_volume_identifier       volume;
    Pcte_volume_accessibility    mounted;
} Pcte_volume_info;
```

(20)
```
typedef struct {
    Pcte_object_reference    origin;
    Pcte_link_names          links;
} Pcte_link_set_descriptor;
```

(21)
```
typedef struct {
    Pcte_object_reference    origin;
    Pcte_link_references     links;
} Pcte_h_link_set_descriptor;
```

(22)
```
#define PCTE_MAX_EXACT_IDENTIFIER_SIZE PCTE_MAX_KEY_SIZE
```

(23)
```
typedef Pcte_octet
    Pcte_exact_identifier [PCTE_MAX_EXACT_IDENTIFIER_SIZE + 1];
```

(24)
```
#include <Pcte/discretionary.h>
```

9.2 Link operations

```
/* 9.2.1 LINK_CREATE */
```

(1)
```
Pcte_error_type Pcte_link_create (
    Pcte_object_reference    origin,
    Pcte_link_name           new_link,
    Pcte_object_reference    dest,
    Pcte_key                 reverse_key
);
```

(2)
```
Pcte_error_type Pcte_h_link_create (
    Pcte_object_reference    origin,
    Pcte_link_reference      new_link,
    Pcte_object_reference    dest,
    Pcte_key                 reverse_key
);
```

(3) /* The effect of not providing the optional parameter *reverse_key* to the abstract operation */
/* is achieved by specifying **reverse_key** as NULL. */

```
/* 9.2.2 LINK_DELETE */
```

(4)
```
Pcte_error_type Pcte_link_delete (
    Pcte_object_reference    origin,
    Pcte_link_name           link
);
```

(5)
```
Pcte_error_type Pcte_h_link_delete (
    Pcte_object_reference   origin,
    Pcte_link_reference     link
);
```

```
/* 9.2.3 LINK_DELETE_ATTRIBUTE */
```

(6)
```
Pcte_error_type Pcte_link_delete_attribute (
    Pcte_object_reference   origin,
    Pcte_link_name          link,
    Pcte_attribute_name     attribute
);
```

(7)
```
Pcte_error_type Pcte_h_link_delete_attribute (
    Pcte_object_reference       origin,
    Pcte_link_reference         link,
    Pcte_attribute_reference    attribute
);
```

```
/* 9.2.4 LINK_GET_ATTRIBUTE */
```

(8)
```
Pcte_error_type Pcte_link_get_attribute (
    Pcte_object_reference   origin,
    Pcte_link_name          link,
    Pcte_attribute_name     attribute,
    Pcte_attribute_value    *value
);
```

(9)
```
Pcte_error_type Pcte_h_link_get_attribute (
    Pcte_object_reference       origin,
    Pcte_link_reference         link,
    Pcte_attribute_reference    attribute,
    Pcte_attribute_value        *value
);
```

```
/* 9.2.5 LINK_GET_DESTINATION_VOLUME */
```

(10)
```
Pcte_error_type Pcte_link_get_destination_volume (
    Pcte_object_reference   origin,
    Pcte_link_name          link,
    Pcte_volume_info        *volume_info
);
```

(11)
```
Pcte_error_type Pcte_h_link_get_destination_volume (
    Pcte_object_reference   origin,
    Pcte_link_reference     link,
    Pcte_volume_info        *volume_info
);
```

```
/* 9.2.6 LINK_GET_KEY */
```

(12)
```
Pcte_error_type Pcte_link_get_key (
    Pcte_object_reference   origin,
    Pcte_link_name          link,
    Pcte_key                key
);
```

(13)
```
Pcte_error_type Pcte_h_link_get_key (
    Pcte_object_reference   origin,
    Pcte_link_reference     link,
    Pcte_key                key
);
```

/* 9.2.7 LINK_GET_REVERSE */

(14)
```
Pcte_error_type Pcte_link_get_reverse (
    Pcte_object_reference   origin,
    Pcte_link_name          link,
    Pcte_link_name          reverse_link,
    Pcte_object_reference   *dest
);
```

(15)
```
Pcte_error_type Pcte_h_link_get_reverse (
    Pcte_object_reference   origin,
    Pcte_link_reference     link,
    Pcte_link_reference     *reverse_link,
    Pcte_object_reference   *dest
);
```

(16) /* If the abstract operation returns no value in *reverse_link* then **reverse_link** is set to NULL */
/* for Pcte_h_link_get_reverse and is set to a string of zero length for Pcte_link_get_reverse. */

/* 9.2.8 LINK_GET_SEVERAL_ATTRIBUTES */

(17)
```
Pcte_error_type Pcte_link_get_attributes_in_working_schema (
    Pcte_object_reference         origin,
    Pcte_link_name                link,
    Pcte_attribute_assignments    *values
);
```

(18)
```
Pcte_error_type Pcte_h_link_get_attributes_in_working_schema (
    Pcte_object_reference         origin,
    Pcte_link_reference           link,
    Pcte_h_attribute_assignments  *values
);
```

(19)
```
Pcte_error_type Pcte_link_get_attributes_of_types (
    Pcte_object_reference         origin,
    Pcte_link_name                link,
    Pcte_attribute_names          attributes,
    Pcte_attribute_assignments    *values
);
```

(20)
```
Pcte_error_type Pcte_h_link_get_attributes_of_types (
    Pcte_object_reference         origin,
    Pcte_link_reference           link,
    Pcte_attribute_references     attributes,
    Pcte_h_attribute_assignments  *values
);
```

(21) /* The effect of specifying *attributes* as VISIBLE_ATTRIBUTE_TYPES to the abstract */
/* operation is achieved by the operation Pcte_link_get_attributes_in_working_schema. */

```
/*  The effect of specifying attributes as a set of attribute designators to the abstract   */
/*  operation is achieved by the operation Pcte_link_get_attributes_of_types.               */

/* 9.2.9 LINK_REPLACE */
```

(22)
```
Pcte_error_type Pcte_link_replace (
    Pcte_object_reference       origin,
    Pcte_link_name              link,
    Pcte_object_reference       new_origin,
    Pcte_link_name              new_link,
    Pcte_key                    new_reverse_key
);
```

(23)
```
Pcte_error_type Pcte_h_link_replace (
    Pcte_object_reference       origin,
    Pcte_link_reference         link,
    Pcte_object_reference       new_origin,
    Pcte_link_reference         new_link,
    Pcte_key                    new_reverse_key
);

/* 9.2.10 LINK_RESET_ATTRIBUTE */
```

(24)
```
Pcte_error_type Pcte_link_reset_attribute (
    Pcte_object_reference       origin,
    Pcte_link_name              link,
    Pcte_attribute_name         attribute
);
```

(25)
```
Pcte_error_type Pcte_h_link_reset_attribute (
    Pcte_object_reference       origin,
    Pcte_link_reference         link,
    Pcte_attribute_reference    attribute
);

/* 9.2.11 LINK_SET_ATTRIBUTE */
```

(26)
```
Pcte_error_type Pcte_link_set_attribute (
    Pcte_object_reference       origin,
    Pcte_link_name              link,
    Pcte_attribute_name         attribute,
    Pcte_attribute_value        *value
);
```

(27)
```
Pcte_error_type Pcte_h_link_set_attribute (
    Pcte_object_reference       origin,
    Pcte_link_reference         link,
    Pcte_attribute_reference    attribute,
    Pcte_attribute_value        *value
);
```

```
/* 9.2.12 LINK_SET_SEVERAL_ATTRIBUTES */
```

(28)
```
Pcte_error_type Pcte_link_set_several_attributes (
    Pcte_object_reference           origin,
    Pcte_link_name                  link,
    Pcte_attribute_assignments      attributes
);
```

(29)
```
Pcte_error_type Pcte_h_link_set_several_attributes (
    Pcte_object_reference           origin,
    Pcte_link_reference             link,
    Pcte_h_attribute_assignments    attributes
);
```

9.3 Object operations

```
/* 9.3.1 OBJECT_CHECK_TYPE */
```

(1)
```
Pcte_error_type Pcte_object_check_type (
    Pcte_object_reference   object,
    Pcte_type_name          type2,
    Pcte_type_ancestry      *relation
);
```

(2)
```
Pcte_error_type Pcte_h_object_check_type (
    Pcte_object_reference   object,
    Pcte_type_reference     type2,
    Pcte_type_ancestry      *relation
);
```

```
/* 9.3.2 OBJECT_CONVERT */
```

(3)
```
Pcte_error_type Pcte_object_convert (
    Pcte_object_reference   object,
    Pcte_type_name          type
);
```

(4)
```
Pcte_error_type Pcte_h_object_convert (
    Pcte_object_reference   object,
    Pcte_type_reference     type
);
```

```
/* 9.3.3 OBJECT_COPY */
```

(5)
```
Pcte_error_type Pcte_object_copy (
    Pcte_object_reference       object,
    Pcte_object_reference       new_origin,
    Pcte_link_name              new_link,
    Pcte_key                    reverse_key,
    Pcte_object_reference       on_same_volume_as,
    Pcte_atomic_access_rights   *access_mask,
    Pcte_object_reference       *new_object
);
```

```
(6)     Pcte_error_type Pcte_h_object_copy (
            Pcte_object_reference           object,
            Pcte_object_reference           new_origin,
            Pcte_link_reference             new_link,
            Pcte_key                        reverse_key,
            Pcte_object_reference           on_same_volume_as,
            Pcte_atomic_access_rights       *access_mask,
            Pcte_object_reference           *new_object
        );
```

(7) /* The effect of not providing the optional parameter *reverse_key* to the abstract operation is achieved by specifying **reverse_key** as NULL. The effect of not providing the optional parameter *on_same_volume_as* to the abstract operation is achieved by specifying **on_same_volume_as** as Pcte_null_object_reference. */

/* 9.3.4 OBJECT_CREATE */

```
(8)     Pcte_error_type Pcte_object_create (
            Pcte_type_name                  type,
            Pcte_object_reference           new_origin,
            Pcte_link_name                  new_link,
            Pcte_key                        reverse_key,
            Pcte_object_reference           on_same_volume_as,
            Pcte_atomic_access_rights       *access_mask,
            Pcte_object_reference           *new_object
        );

(9)     Pcte_error_type Pcte_h_object_create (
            Pcte_type_reference             type,
            Pcte_object_reference           new_origin,
            Pcte_link_reference             new_link,
            Pcte_key                        reverse_key,
            Pcte_object_reference           on_same_volume_as,
            Pcte_atomic_access_rights       *access_mask,
            Pcte_object_reference           *new_object
        );
```

(10) /* The effect of not providing the optional parameter *reverse_key* to the abstract operation is achieved by specifying **reverse_key** as NULL. The effect of not providing the optional parameter *on_same_volume_as* to the abstract operation is achieved by specifying **on_same_volume_as** as Pcte_null_object_reference. */

/* 9.3.5 OBJECT_DELETE */

```
(11)    Pcte_error_type Pcte_object_delete (
            Pcte_object_reference   origin,
            Pcte_link_name          link
        );

(12)    Pcte_error_type Pcte_h_object_delete (
            Pcte_object_reference   origin,
            Pcte_link_reference     link
        );
```

/* 9.3.6 OBJECT_DELETE_ATTRIBUTE */

(13)
```
Pcte_error_type Pcte_object_delete_attribute (
    Pcte_object_reference    object,
    Pcte_attribute_name      attribute
);
```

(14)
```
Pcte_error_type Pcte_h_object_delete_attribute (
    Pcte_object_reference       object,
    Pcte_attribute_reference    attribute
);
```

/* 9.3.7 OBJECT_GET_ATTRIBUTE */

(15)
```
Pcte_error_type Pcte_object_get_attribute (
    Pcte_object_reference    object,
    Pcte_attribute_name      attribute,
    Pcte_attribute_value     *value
);
```

(16)
```
Pcte_error_type Pcte_h_object_get_attribute (
    Pcte_object_reference       object,
    Pcte_attribute_reference    attribute,
    Pcte_attribute_value        *value
);
```

/* 9.3.8 OBJECT_GET_PREFERENCE */

(17)
```
Pcte_error_type Pcte_object_get_preference (
    Pcte_object_reference    object,
    Pcte_key                 key,
    Pcte_type_name           type
);
```

(18)
```
Pcte_error_type Pcte_h_object_get_preference (
    Pcte_object_reference    object,
    Pcte_key                 key,
    Pcte_type_reference      *type
);
```

(19) /* If the abstract operation returns no *key*, **key** is set as a string of zero length. If the abstract */
/* operation returns no *type*, **type** is set to NULL for Pcte_h_object_get_preference and to a */
/* string of zero length for Pcte_object_get_preference. */

/* 9.3.9 OBJECT_GET_SEVERAL_ATTRIBUTES */

(20)
```
Pcte_error_type Pcte_object_get_attributes_in_working_schema (
    Pcte_object_reference         object,
    Pcte_attribute_assignments    *values
);
```

(21)
```
Pcte_error_type Pcte_h_object_get_attributes_in_working_schema (
    Pcte_object_reference           object,
    Pcte_h_attribute_assignments    *values
);
```

```
(22)    Pcte_error_type Pcte_object_get_attributes_of_types (
            Pcte_object_reference           object,
            Pcte_attribute_names            attributes,
            Pcte_attribute_assignments      *values
        );

(23)    Pcte_error_type Pcte_h_object_get_attributes_of_types (
            Pcte_object_reference               object,
            Pcte_attribute_references           attributes,
            Pcte_h_attribute_assignments        *values
        );

(24)    /*  The effect of specifying attributes as VISIBLE_ATTRIBUTE_TYPES to the abstract       */
        /*  operation is achieved by the operation Pcte_object_get_attributes_in_working_schema.  */
        /*  The effect of specifying attributes as a set of attribute designators to the abstract */
        /*  operation is achieved by the operation Pcte_object_get_attributes_of_types.           */

        /* 9.3.10 OBJECT_GET_TYPE */

(25)    Pcte_error_type Pcte_object_get_type (
            Pcte_object_reference   object,
            Pcte_type_name          type
        );

(26)    Pcte_error_type Pcte_h_object_get_type (
            Pcte_object_reference   object,
            Pcte_type_reference     *type
        );

        /* 9.3.11 OBJECT_IS_COMPONENT */

(27)    Pcte_error_type Pcte_object_is_component (
            Pcte_object_reference   object1,
            Pcte_object_reference   object2,
            Pcte_boolean            *value
        );

        /* 9.3.12 OBJECT_LIST_LINKS */

(28)    Pcte_error_type Pcte_object_list_all_links (
            Pcte_object_reference       origin,
            Pcte_link_scope             extent,
            Pcte_object_scope           scope,
            Pcte_categories             categories,
            Pcte_link_set_descriptors   *links
        );

(29)    Pcte_error_type Pcte_h_object_list_all_links (
            Pcte_object_reference           origin,
            Pcte_link_scope                 extent,
            Pcte_object_scope               scope,
            Pcte_categories                 categories,
            Pcte_h_link_set_descriptors     *links
        );
```

(30)
```
/*  The effect of specifying visibility as ALL_LINK_TYPES is achieved by the operation      */
/*  Pcte_object_list_all_links.  For Pcte_object_list_all_links, all the link type names in the  */
/*  returned link names are type identifiers.                                               */
```

(31)
```
Pcte_error_type Pcte_object_list_links_in_working_schema (
    Pcte_object_reference       origin,
    Pcte_link_scope             extent,
    Pcte_object_scope           scope,
    Pcte_categories             categories,
    Pcte_link_set_descriptors   *links
);
```

(32)
```
Pcte_error_type Pcte_h_object_list_links_in_working_schema (
    Pcte_object_reference           origin,
    Pcte_link_scope                 extent,
    Pcte_object_scope               scope,
    Pcte_categories                 categories,
    Pcte_h_link_set_descriptors     *links
);
```

(33)
```
/*  The effect of specifying visibility as VISIBLE_TYPES is achieved by the operation       */
/*  Pcte_object_list_links_in_working_schema.                                               */
```

(34)
```
Pcte_error_type Pcte_object_list_links_of_types (
    Pcte_object_reference       origin,
    Pcte_link_scope             extent,
    Pcte_object_scope           scope,
    Pcte_type_names             types,
    Pcte_link_set_descriptors   *links
);
```

(35)
```
Pcte_error_type Pcte_h_object_list_links_of_types (
    Pcte_object_reference           origin,
    Pcte_link_scope                 extent,
    Pcte_object_scope               scope,
    Pcte_type_references            types,
    Pcte_h_link_set_descriptors     *links
);
```

(36)
```
/*  The effect of specifying visibility as link type nominators is achieved by the operation  */
/*  Pcte_object_list_links_of_types.                                                        */
```

```
/* 9.3.13 OBJECT_LIST_VOLUMES */
```

(37)
```
Pcte_error_type Pcte_object_list_volumes (
    Pcte_object_reference   object,
    Pcte_volume_infos       *volumes
);
```

```
/* 9.3.14 OBJECT_MOVE */
```

(38)
```
Pcte_error_type Pcte_object_move (
    Pcte_object_reference   object,
    Pcte_object_reference   on_same_volume_as,
    Pcte_object_scope       scope
);
```

```
        /* 9.3.15 OBJECT_RESET_ATTRIBUTE */

(39)    Pcte_error_type Pcte_object_reset_attribute (
            Pcte_object_reference    object,
            Pcte_attribute_name      attribute
        );

(40)    Pcte_error_type Pcte_h_object_reset_attribute (
            Pcte_object_reference       object,
            Pcte_attribute_reference    attribute
        );

        /* 9.3.16 OBJECT_SET_ATTRIBUTE */

(41)    Pcte_error_type Pcte_object_set_attribute (
            Pcte_object_reference    object,
            Pcte_attribute_name      attribute,
            Pcte_attribute_value     *value
        );

(42)    Pcte_error_type Pcte_h_object_set_attribute (
            Pcte_object_reference       object,
            Pcte_attribute_reference    attribute,
            Pcte_attribute_value        *value
        );

        /* 9.3.17 OBJECT_SET_PREFERENCE */

(43)    Pcte_error_type Pcte_object_set_preference (
            Pcte_object_reference    object,
            Pcte_type_name           type,
            Pcte_key                 key
        );

(44)    Pcte_error_type Pcte_h_object_set_preference (
            Pcte_object_reference    object,
            Pcte_type_reference      type,
            Pcte_key                 key
        );
```

(45) /* The effect of not providing the optional parameter *type* to the abstract operation is */
/* achieved by specifying **type** as NULL. The effect of not providing the optional */
/* parameter *key* to the abstract operation is achieved by specifying **key** as NULL. */

```
        /* 9.3.18 OBJECT_SET_SEVERAL_ATTRIBUTES */

(46)    Pcte_error_type Pcte_object_set_several_attributes (
            Pcte_object_reference          object,
            Pcte_attribute_assignments     attributes
        );

(47)    Pcte_error_type Pcte_h_object_set_several_attributes (
            Pcte_object_reference            object,
            Pcte_h_attribute_assignments     attributes
        );
```

/* 9.3.19 OBJECT_SET_TIME_ATTRIBUTES */

(48)
```
Pcte_error_type Pcte_object_set_time_attributes (
    Pcte_object_reference   object,
    Pcte_time               last_access,
    Pcte_time               last_modification,
    Pcte_object_scope       scope
);
```

(49)
/* The effect of not providing the optional parameters *last_access* or *last_modification* to the */
/* abstract operation is achieved by specifying **last_access** or **last_modification** as */
/* Pcte_null_time. */

/* 9.3.20 VOLUME_LIST_OBJECTS */

(50)
```
Pcte_error_type Pcte_volume_list_objects (
    Pcte_object_reference   volume,
    Pcte_type_names         types,
    Pcte_object_references  *objects
);
```

(51)
```
Pcte_error_type Pcte_h_volume_list_objects (
    Pcte_object_reference   volume,
    Pcte_type_references    types,
    Pcte_object_references  *objects
);
```

9.4 Version operations

/* 9.4.1 VERSION_ADD_PREDECESSOR */

(1)
```
Pcte_error_type Pcte_version_add_predecessor (
    Pcte_object_reference       version,
    Pcte_object_reference       new_predecessor
);
```

/* 9.4.2 VERSION_IS_CHANGED */

(2)
```
Pcte_error_type Pcte_version_is_changed (
    Pcte_object_reference   version,
    Pcte_key                predecessor,
    Pcte_boolean            *changed
);
```

/* 9.4.3 VERSION_REMOVE */

(3)
```
Pcte_error_type Pcte_version_remove (
    Pcte_object_reference       version
);
```

/* 9.4.4 VERSION_REMOVE_PREDECESSOR */

(4)
```
Pcte_error_type Pcte_version_remove_predecessor (
    Pcte_object_reference       version,
    Pcte_object_reference       predecessor
);
```

/* 9.4.5 VERSION_REVISE */

(5)
```
Pcte_error_type Pcte_version_revise (
    Pcte_object_reference        version,
    Pcte_object_reference        new_origin,
    Pcte_link_name               new_link,
    Pcte_object_reference        on_same_volume_as,
    Pcte_atomic_access_rights    *access_mask,
    Pcte_object_reference        *new_version
);
```

(6)
```
Pcte_error_type Pcte_h_version_revise (
    Pcte_object_reference        version,
    Pcte_object_reference        new_origin,
    Pcte_link_reference          new_link,
    Pcte_object_reference        on_same_volume_as,
    Pcte_atomic_access_rights    *access_mask,
    Pcte_object_reference        *new_version
);
```

(7) /* The effect of not providing the optional parameter *on_same_volume_as* to the abstract */
/* operation is achieved by specifying **on_same_volume_as** as Pcte_null_object_reference. */

/* 9.4.6 VERSION_SNAPSHOT */

(8)
```
Pcte_error_type Pcte_version_snapshot (
    Pcte_object_reference        version,
    Pcte_object_reference        new_origin,
    Pcte_link_name               new_link,
    Pcte_object_reference        on_same_volume_as,
    Pcte_atomic_access_rights    *access_mask,
    Pcte_object_reference        *new_version
);
```

(9)
```
Pcte_error_type Pcte_h_version_snapshot (
    Pcte_object_reference        version,
    Pcte_object_reference        new_origin,
    Pcte_link_reference          new_link,
    Pcte_object_reference        on_same_volume_as,
    Pcte_atomic_access_rights    *access_mask,
    Pcte_object_reference        *new_version
);
```

(10) /* The effect of not providing the optional parameter *new_link_and_origin* to the abstract */
/* operation is achieved by specifying **new_link** as NULL and new_origin as */
/* Pcte_null_object_reference. The effect of not providing the optional parameter */
/* *on_same_volume_as* to the abstract operation is achieved by specifying */
/* **on_same_volume_as** as Pcte_null_object_reference. */

```
/* 9.4.7 VERSION_TEST_ANCESTRY */
```

(11)
```
Pcte_error_type Pcte_version_test_ancestry (
    Pcte_object_reference        version1,
    Pcte_object_reference        version2,
    Pcte_version_relation        *ancestry
);
```

```
/* 9.4.8 VERSION_TEST_DESCENT */
```

(12)
```
Pcte_error_type Pcte_version_test_descent (
    Pcte_object_reference        version1,
    Pcte_object_reference        version2,
    Pcte_version_relation        *descent
);
```

(13)
```
#endif /* !PCTE_OMS_INCLUDED */
```

10 Schema management

(1)
```
/* The header <Pcte/sms.h> */
```

(2)
```
#ifndef PCTE_SMS_INCLUDED
#define PCTE_SMS_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
#include <Pcte/sequences.h>
#include <Pcte/oms.h>
```

10.1 Schema management datatypes

(1)
```
typedef enum {
    PCTE_CREATE_MODE        =  1<<0,
    PCTE_DELETE_MODE        =  1<<1,
    PCTE_READ_MODE          =  1<<2,
    PCTE_WRITE_MODE         =  1<<3,
    PCTE_NAVIGATE_MODE      =  1<<4
} Pcte_definition_mode_value;
```

(2)
```
typedef Pcte_natural Pcte_definition_mode_values;
```

(3)
```
typedef enum {
    PCTE_DUPLICATED, PCTE_NOT_DUPLICATED
} Pcte_duplication;
```

(4)
```
typedef enum {
    PCTE_SHARABLE, PCTE_EXCLUSIVE
} Pcte_exclusiveness;
```

(5)
```
typedef enum {
    PCTE_ATOMIC_STABLE, PCTE_COMPOSITE_STABLE, PCTE_NOT_STABLE
} Pcte_stability;
```

(6)
```
typedef enum {
    PCTE_NO_CONTENTS, PCTE_FILE_TYPE, PCTE_PIPE_TYPE,
    PCTE_DEVICE_TYPE, PCTE_AUDIT_FILE_TYPE,
    PCTE_ACCOUNTING_LOG_TYPE
} Pcte_contents_type;
```

(7)
```
/*  Pcte_contents_type corresponds to the PCTE datatype Contents_type.  The value   */
/*  PCTE_NO_CONTENTS corresponds to absence of a Contents_type result from          */
/*  SDS_GET_OBJECT_TYPE_PROPERTIES and                                              */
/*  WS_GET_OBJECT_TYPE_PROPERTIES.                                                  */
```

(8)
```
typedef struct {
    Pcte_category        category;
    Pcte_stability       stability;
    Pcte_exclusiveness   exclusiveness;
    Pcte_duplication     duplication;
} Pcte_link_flags;
```

(9)
```
typedef struct {
    Pcte_link_flags   link_type_flag;
    Pcte_natural      lower_bound, upper_bound;
} Pcte_link_type_properties;
```

(10)
```
/*  Pcte_link_type_properties corresponds to a number of parameter types in         */
/*  SDS_CREATE_RELATIONSHIP_TYPE, and to a number of result types of                */
/*  SDS_GET_LINK_TYPE_PROPERTIES and                                                */
/*  WS_GET_LINK_TYPE_PROPERTIES.                                                    */
```

(11)
```
typedef enum {
    PCTE_OBJECT, PCTE_OBJECT_ALL,
    PCTE_LINK_KEY, PCTE_LINK_NON_KEY
} Pcte_attribute_scan_kind;
```

(12)
```
typedef enum {
    PCTE_ORIGIN, PCTE_ORIGIN_ALL, PCTE_DESTINATION,
    PCTE_DESTINATION_ALL, PCTE_KEY, PCTE_NON_KEY
} Pcte_link_scan_kind;
```

(13)
```
typedef enum {
    PCTE_CHILD, PCTE_DESCENDANT, PCTE_PARENT, PCTE_ANCESTOR,
    PCTE_ATTRIBUTE, PCTE_ATTRIBUTE_ALL, PCTE_LINK_ORIGIN,
    PCTE_LINK_ORIGIN_ALL, PCTE_LINK_DESTINATION,
    PCTE_LINK_DESTINATION_ALL
} Pcte_object_scan_kind;
```

(14)
```
typedef enum {
    PCTE_OBJECT_TYPE, PCTE_LINK_TYPE, PCTE_ATTRIBUTE_TYPE,
    PCTE_ENUMERAL_TYPE
} Pcte_type_kind;
```

(15)
```
#define PCTE_MAX_ENUMERAL_TYPE_IMAGE_SIZE PCTE_MAX_NAME_SIZE
```

(16)
```
typedef Pcte_octet
Pcte_enumeral_type_image [PCTE_MAX_ENUMERAL_TYPE_IMAGE_SIZE + 1];
```

10.2 Update operations

```
     /* 10.2.1 SDS_ADD_DESTINATION */

(1)  Pcte_error_type Pcte_sds_add_destination (
         Pcte_object_reference      sds,
         Pcte_type_name_in_sds      link_type,
         Pcte_type_name_in_sds      object_type
     );

     /* 10.2.2 SDS_APPLY_ATTRIBUTE_TYPE */

(2)  Pcte_error_type Pcte_sds_apply_attribute_type (
         Pcte_object_reference      sds,
         Pcte_type_name_in_sds      attribute_type,
         Pcte_type_name_in_sds      type
     );

     /* 10.2.3 SDS_APPLY_LINK_TYPE */

(3)  Pcte_error_type Pcte_sds_apply_link_type (
         Pcte_object_reference      sds,
         Pcte_type_name_in_sds      link_type,
         Pcte_type_name_in_sds      object_type
     );

     /* 10.2.4 SDS_CREATE_BOOLEAN_ATTRIBUTE_TYPE */

(4)  Pcte_error_type Pcte_sds_create_boolean_attribute_type (
         Pcte_object_reference      sds,
         Pcte_name                  local_name,
         Pcte_boolean               initial_value,
         Pcte_duplication           duplication,
         Pcte_type_name_in_sds      new_type
     );
```

(5) /* The effect of not providing the optional parameter *local_name* to the abstract operation is */
/* achieved by specifying **local_name** as NULL. The effect of not providing the optional */
/* parameter *initial_value* to the abstract operation is achieved by specifying **initial_value** as */
/* PCTE_FALSE. */

```
     /* 10.2.5 SDS_CREATE_DESIGNATION_LINK_TYPE */

(6)  Pcte_error_type Pcte_sds_create_designation_link_type (
         Pcte_object_reference      object,
         Pcte_name                  local_name,
         Pcte_natural               lower_bound,
         Pcte_natural               upper_bound,
         Pcte_duplication           duplication,
         Pcte_key_types_in_sds      key_types,
         Pcte_type_name_in_sds      new_type
     );
```

(7) /* The effect of not providing the optional parameter *local_name* to the abstract operation is */
/* achieved by specifying **local_name** as NULL. The effect of not providing the optional */

```
/*  parameter upper_bound to the abstract operation is achieved by specifying upper_bound  */
/*  as 0.                                                                                  */
```

/* 10.2.6 SDS_CREATE_ENUMERAL_TYPE */

(8)
```
Pcte_error_type Pcte_sds_create_enumeral_type (
    Pcte_object_reference    sds,
    Pcte_name                local_name,
    Pcte_type_name_in_sds    new_type
);
```

(9)
```
/*  The effect of not providing the optional parameter local_name to the abstract operation is  */
/*  achieved by specifying local_name as NULL.                                                  */
```

/* 10.2.7 SDS_CREATE_ENUMERATION_ATTRIBUTE_TYPE */

(10)
```
Pcte_error_type Pcte_sds_create_enumeration_attribute_type (
    Pcte_object_reference    sds,
    Pcte_name                local_name,
    Pcte_type_names_in_sds   values,
    Pcte_duplication         duplication,
    Pcte_natural             initial_value,
    Pcte_type_name_in_sds    new_type
);
```

(11)
```
/*  The effect of not providing the optional parameter local_name to the abstract operation is  */
/*  achieved by specifying local_name as NULL.  The effect of not providing the optional        */
/*  parameter initial_value to the abstract operation is achieved by specifying initial_value   */
/*  as 0.                                                                                       */
```

/* 10.2.8 SDS_CREATE_FLOAT_ATTRIBUTE_TYPE */

(12)
```
Pcte_error_type Pcte_sds_create_float_attribute_type (
    Pcte_object_reference    sds,
    Pcte_name                local_name,
    Pcte_float               initial_value,
    Pcte_duplication         duplication,
    Pcte_type_name_in_sds    new_type
);
```

(13)
```
/*  The effect of not providing the optional parameter local_name to the abstract operation     */
/*  is achieved by specifying local_name as NULL.  The effect of not providing the              */
/*  optional parameter initial_value to the abstract operation is achieved by specifying        */
/*  initial_value as 0.0.                                                                       */
```

/* 10.2.9 SDS_CREATE_INTEGER_ATTRIBUTE_TYPE */

(14)
```
Pcte_error_type Pcte_sds_create_integer_attribute_type (
    Pcte_object_reference    sds,
    Pcte_name                local_name,
    Pcte_integer             initial_value,
    Pcte_duplication         duplication,
    Pcte_type_name_in_sds    new_type
);
```

(15)
```
/*  The effect of not providing the optional parameter local_name to the abstract operation is  */
/*  achieved by specifying local_name as NULL.  The effect of not providing the optional        */
/*  parameter initial_value to the abstract operation is achieved by specifying initial_value   */
/*  as 0.                                                                                        */
```

/* 10.2.10 SDS_CREATE_NATURAL_ATTRIBUTE_TYPE */

(16)
```
Pcte_error_type Pcte_sds_create_natural_attribute_type (
    Pcte_object_reference       sds,
    Pcte_name                   local_name,
    Pcte_natural                initial_value,
    Pcte_duplication            duplication,
    Pcte_type_name_in_sds       new_type
);
```

(17)
```
/*  The effect of not providing the optional parameter local_name to the abstract operation is  */
/*  achieved by specifying local_name as NULL.  The effect of not providing the optional        */
/*  parameter initial_value to the abstract operation is achieved by specifying initial_value   */
/*  as 0.                                                                                        */
```

/* 10.2.11 SDS_CREATE_OBJECT_TYPE */

(18)
```
Pcte_error_type Pcte_sds_create_object_type (
    Pcte_object_reference       sds,
    Pcte_name                   local_name,
    Pcte_type_names_in_sds      parents,
    Pcte_type_name_in_sds       new_type
);
```

(19)
```
/*  The effect of not providing the optional parameter local_name to the abstract operation is  */
/*  achieved by specifying local_name as NULL.                                                   */
```

/* 10.2.12 SDS_CREATE_RELATIONSHIP_TYPE */

(20)
```
Pcte_error_type Pcte_sds_create_relationship_type (
    Pcte_object_reference       sds,
    Pcte_name                   forward_local_name,
    Pcte_link_type_properties   *forward_properties,
    Pcte_key_types_in_sds       forward_key_types,
    Pcte_name                   reverse_local_name,
    Pcte_link_type_properties   *reverse_properties,
    Pcte_key_types_in_sds       reverse_key_types,
    Pcte_type_name_in_sds       forward_type,
    Pcte_type_name_in_sds       reverse_type
);
```

(21)
```
/*  The effect of not providing the optional parameter forward_local_name to the abstract       */
/*  operation is achieved by specifying forward_local_name as NULL.  The effect of not          */
/*  providing the optional parameter reverse_local_name to the abstract operation is achieved   */
/*  by specifying reverse_local_name as NULL.  The effect of not providing the optional         */
/*  parameter forward_upper_bound to the abstract operation is achieved by specifying           */
/*  forward_properties.upper_bound as 0.  The effect of not providing the optional               */
/*  parameter reverse_upper_bound to the abstract operation is achieved by specifying           */
/*  reverse_properties.upper_bound as 0.                                                         */
```

/* 10.2.13 SDS_CREATE_STRING_ATTRIBUTE_TYPE */

(22)
```
Pcte_error_type Pcte_sds_create_string_attribute_type (
    Pcte_object_reference       sds,
    Pcte_name                   local_name,
    Pcte_string                 *initial_value,
    Pcte_duplication            duplication,
    Pcte_type_name_in_sds       new_type
);
```

(23) /* The effect of not providing the optional parameter *local_name* to the abstract operation is */
/* achieved by specifying **local_name** as NULL. The effect of not providing the optional */
/* parameter *initial_value* to the abstract operation is achieved by specifying **initial_value** */
/* as NULL. */

/* 10.2.14 SDS_CREATE_TIME_ATTRIBUTE_TYPE */

(24)
```
Pcte_error_type Pcte_sds_create_time_attribute_type (
    Pcte_object_reference       sds,
    Pcte_name                   local_name,
    Pcte_time                   initial_value,
    Pcte_duplication            duplication,
    Pcte_type_name_in_sds       new_type
);
```

(25) /* The effect of not providing the optional parameter *local_name* to the abstract operation is */
/* achieved by specifying **local_name** as NULL. The effect of not providing the optional */
/* parameter *initial_value* to the abstract operation is achieved by specifying **initial_value** */
/* as Pcte_reference_time. */

/* 10.2.15 SDS_IMPORT_ATTRIBUTE_TYPE */

(26)
```
Pcte_error_type Pcte_sds_import_attribute_type (
    Pcte_object_reference       to_sds,
    Pcte_object_reference       from_sds,
    Pcte_type_name_in_sds       type,
    Pcte_name                   local_name
);
```

(27) /* The effect of not providing the optional parameter *local_name* to the abstract operation is */
/* achieved by specifying **local_name** as NULL. */

/* 10.2.16 SDS_GET_NAME */

(28)
```
Pcte_error_type Pcte_sds_get_name (
    Pcte_object_reference   sds,
    Pcte_name               name
);
```

```
/* 10.2.17 SDS_IMPORT_ENUMERAL_TYPE */
```

(29)
```
Pcte_error_type Pcte_sds_import_enumeral_type (
    Pcte_object_reference      to_sds,
    Pcte_object_reference      from_sds,
    Pcte_type_name_in_sds      type,
    Pcte_name                  local_name
);
```

(30) /* The effect of not providing the optional parameter *local_name* to the abstract operation is */
/* achieved by specifying **local_name** as NULL. */

```
/* 10.2.18 SDS_IMPORT_LINK_TYPE */
```

(31)
```
Pcte_error_type Pcte_sds_import_link_type (
    Pcte_object_reference      to_sds,
    Pcte_object_reference      from_sds,
    Pcte_type_name_in_sds      type,
    Pcte_name                  local_name
);
```

(32) /* The effect of not providing the optional parameter *local_name* to the abstract operation is */
/* achieved by specifying **local_name** as NULL. */

```
/* 10.2.19 SDS_IMPORT_OBJECT_TYPE */
```

(33)
```
Pcte_error_type Pcte_sds_import_object_type (
    Pcte_object_reference      to_sds,
    Pcte_object_reference      from_sds,
    Pcte_type_name_in_sds      type,
    Pcte_name                  local_name
);
```

(34) /* The effect of not providing the optional parameter *local_name* to the abstract operation is */
/* achieved by specifying **local_name** as NULL. */

```
/* 10.2.20 SDS_INITIALIZE */
```

(35)
```
Pcte_error_type Pcte_sds_initialize (
    Pcte_object_reference  sds,
    Pcte_name              name
);
```

```
/* 10.2.21 SDS_REMOVE */
```

(36)
```
Pcte_error_type Pcte_sds_remove (
    Pcte_object_reference  sds
);
```

```
/* 10.2.22 SDS_REMOVE_DESTINATION */
```

(37)
```
Pcte_error_type Pcte_sds_remove_destination (
    Pcte_object_reference      sds,
    Pcte_type_name_in_sds      link_type,
    Pcte_type_name_in_sds      object_type
);
```

```
        /* 10.2.23 SDS_REMOVE_TYPE */

(38)    Pcte_error_type Pcte_sds_remove_type (
            Pcte_object_reference       sds,
            Pcte_type_name_in_sds       type
        );

        /* 10.2.24 SDS_SET_ENUMERAL_TYPE_IMAGE */

(39)    Pcte_error_type Pcte_sds_set_enumeral_type_image (
            Pcte_object_reference           sds,
            Pcte_type_name_in_sds           type,
            Pcte_enumeral_type_image        image
        );

(40)    /*  The effect of not providing the optional parameter image to the abstract operation is     */
        /*  achieved by specifying image as NULL.

        /* 10.2.25 SDS_SET_TYPE_MODES */

(41)    Pcte_error_type Pcte_sds_set_usage_mode (
            Pcte_object_reference           sds,
            Pcte_type_name_in_sds           type,
            Pcte_definition_mode_values     usage_mode
        );

(42)    Pcte_error_type Pcte_sds_set_export_mode (
            Pcte_object_reference           sds,
            Pcte_type_name_in_sds           type,
            Pcte_definition_mode_values     export_mode
        );

(43)    /*  The effect of not providing the optional parameter export_mode is obtained by calling     */
        /*  Pcte_sds_set_usage_mode.  The effect of not providing the optional parameter              */
        /*  usage_mode is obtained by calling Pcte_sds_set_export_mode.  The effect of providing      */
        /*  both optional parameters usage_mode and export_mode is obtained by calling                */
        /*  Pcte_sds_set_usage_mode and Pcte_sds_set_export_mode in sequence.  As an operation        */
        /*  call with neither optional parameter has no effect, no means for making such a call is    */
        /*  provided.                                                                                 */

        /* 10.2.26 SDS_SET_TYPE_NAME */

(44)    Pcte_error_type Pcte_sds_set_type_name (
            Pcte_object_reference       sds,
            Pcte_type_name_in_sds       type,
            Pcte_name                   local_name
        );

(45)    /*  The effect of not providing the optional parameter local_name to the abstract operation is */
        /*  achieved by specifying local_name as NULL.                                                 */
```

/* 10.2.27 SDS_UNAPPLY_ATTRIBUTE_TYPE */

(46)
```
Pcte_error_type Pcte_sds_unapply_attribute_type (
    Pcte_object_reference      sds,
    Pcte_type_name_in_sds      attribute_type,
    Pcte_type_name_in_sds      type
);
```

/* 10.2.28 SDS_UNAPPLY_LINK_TYPE */

(47)
```
Pcte_error_type Pcte_sds_unapply_link_type (
    Pcte_object_reference      sds,
    Pcte_type_name_in_sds      link_type,
    Pcte_type_name_in_sds      object_type
);
```

10.3 Usage operations

/* 10.3.1 SDS_GET_ATTRIBUTE_TYPE_PROPERTIES */

(1)
```
Pcte_error_type Pcte_sds_get_attribute_type_properties (
    Pcte_object_reference                  sds,
    Pcte_type_name_in_sds                  type,
    Pcte_duplication                       *duplication,
    Pcte_value_type                        *value_type,
    Pcte_enumeration_value_type_in_sds     *enumeration_value_type,
    Pcte_attribute_value                   *initial_value
);
```

(2) /* If the abstract operation returns an enumeration value type in *value_type* then **value_type** */
/* is set to PCTE_ENUMERATION_VALUE_TYPE and **enumeration_value_type** contains */
/* the sequence of enumeration value type nominators. */

/* 10.3.2 SDS_GET_ENUMERAL_TYPE_IMAGE */

(3)
```
Pcte_error_type Pcte_sds_get_enumeral_type_image (
    Pcte_object_reference        sds,
    Pcte_type_name_in_sds        enumeral_type,
    Pcte_enumeral_type_image     image
);
```

(4) /* If the abstract operation returns no value in *image*, **image** is returned as a string of zero */
/* length. */

/* 10.3.3 SDS_GET_ENUMERAL_TYPE_POSITION */

(5)
```
Pcte_error_type Pcte_sds_get_enumeral_type_position (
    Pcte_object_reference      sds,
    Pcte_type_name_in_sds      enumeral_type,
    Pcte_type_name_in_sds      attribute_type,
    Pcte_natural               *position
);
```

/* 10.3.4 SDS_GET_LINK_TYPE_PROPERTIES */

(6)
```
Pcte_error_type Pcte_sds_get_link_type_properties (
    Pcte_object_reference        sds,
    Pcte_type_name_in_sds        type,
    Pcte_link_type_properties    *properties,
    Pcte_key_types_in_sds        *key_types,
    Pcte_type_name_in_sds        reverse
);
```

(7) /* The *category*, *lower_bound*, *upper_bound*, *exclusiveness*, *stability*, *duplication*, *key_types* */
/* and *reverse* values are returned in the members of the same names of the */
/* **Pcte_link_type_properties** object pointed to by **properties**. If the abstract operation */
/* returns no value in *reverse*, **reverse** is set to a string of zero length. */

/* 10.3.5 SDS_GET_OBJECT_TYPE_PROPERTIES */

(8)
```
Pcte_error_type Pcte_sds_get_object_type_properties (
    Pcte_object_reference        sds,
    Pcte_type_name_in_sds        type,
    Pcte_contents_type           *contents_type,
    Pcte_type_names_in_sds       *parents,
    Pcte_type_names_in_sds       *children
);
```

(9) /* If the abstract operation returns no value in *contents_type* then contents_type is set to */
/* PCTE_NO_CONTENTS. */

/* 10.3.6 SDS_GET_TYPE_KIND */

(10)
```
Pcte_error_type Pcte_sds_get_type_kind (
    Pcte_object_reference        sds,
    Pcte_type_name_in_sds        type,
    Pcte_type_kind               *type_kind
);
```

/* 10.3.7 SDS_GET_TYPE_MODES */

(11)
```
Pcte_error_type Pcte_sds_get_type_modes (
    Pcte_object_reference          sds,
    Pcte_type_name_in_sds          type,
    Pcte_definition_mode_values    *usage_mode,
    Pcte_definition_mode_values    *export_mode,
    Pcte_definition_mode_values    *max_usage_mode
);
```

/* 10.3.8 SDS_GET_TYPE_NAME */

(12)
```
Pcte_error_type Pcte_sds_get_type_name (
    Pcte_object_reference        sds,
    Pcte_type_name_in_sds        type,
    Pcte_type_name               name
);
```

(13) /* If the abstract operation returns no value in *name*, **name** is returned as a string of zero */
/* length. */

```
      /* 10.3.9 SDS_SCAN_ATTRIBUTE_TYPE */
(14)  Pcte_error_type Pcte_sds_scan_attribute_type (
          Pcte_object_reference       sds,
          Pcte_type_name_in_sds       type,
          Pcte_attribute_scan_kind    scanning_kind,
          Pcte_type_names_in_sds      *types
      );
      /* 10.3.10 SDS_SCAN_ENUMERAL_TYPE */
(15)  Pcte_error_type Pcte_sds_scan_enumeral_type (
          Pcte_object_reference       sds,
          Pcte_type_name_in_sds       type,
          Pcte_type_names_in_sds      *types
      );
      /* 10.3.11 SDS_SCAN_LINK_TYPE */
(16)  Pcte_error_type Pcte_sds_scan_link_type (
          Pcte_object_reference       sds,
          Pcte_type_name_in_sds       type,
          Pcte_link_scan_kind         scanning_kind,
          Pcte_type_names_in_sds      *types
      );
      /* 10.3.12 SDS_SCAN_OBJECT_TYPE */
(17)  Pcte_error_type Pcte_sds_scan_object_type (
          Pcte_object_reference       sds,
          Pcte_type_name_in_sds       type,
          Pcte_object_scan_kind       scanning_kind,
          Pcte_type_names_in_sds      *types
      );
      /* 10.3.13 SDS_SCAN_TYPES */
(18)  Pcte_error_type Pcte_sds_scan_types (
          Pcte_object_reference       sds,
          Pcte_type_kind              kind,
          Pcte_type_names_in_sds      *types
      );
(19)  Pcte_error_type Pcte_sds_scan_all_types (
          Pcte_object_reference       sds,
          Pcte_type_names_in_sds      *types
      );
(20)  /*  The effect of not providing the optional parameter kind to the abstract operation is   */
      /*  achieved by the operation Pcte_sds_scan_all_types.                                     */
```

10.4 Working schema operations

```
/* 10.4.1 WS_GET_ATTRIBUTE_TYPE_PROPERTIES */
```

(1)
```
Pcte_error_type Pcte_ws_get_attribute_type_properties (
    Pcte_type_name               type,
    Pcte_duplication             *duplication,
    Pcte_value_type              *value_type,
    Pcte_enumeration_value_type  *enumeration_value_type,
    Pcte_attribute_value         *initial_value
);
```

(2)
```
Pcte_error_type Pcte_h_ws_get_attribute_type_properties (
    Pcte_type_reference            type,
    Pcte_duplication               *duplication,
    Pcte_value_type                *value_type,
    Pcte_h_enumeration_value_type  *enumeration_value_type,
    Pcte_attribute_value           *initial_value
);
```

```
/* 10.4.2 WS_GET_ENUMERAL_TYPE_IMAGE */
```

(3)
```
Pcte_error_type Pcte_ws_get_enumeral_type_image (
    Pcte_type_name              enumeral_type,
    Pcte_enumeral_type_image    image
);
```

(4)
```
Pcte_error_type Pcte_h_ws_get_enumeral_type_image (
    Pcte_type_reference         enumeral_type,
    Pcte_enumeral_type_image    image
);
```

(5) /* If the abstract operation returns no value in *image*, **image** is returned as a string of zero */
/* length. */

```
/* 10.4.3 WS_GET_ENUMERAL_TYPE_POSITION */
```

(6)
```
Pcte_error_type Pcte_ws_get_enumeral_type_position (
    Pcte_type_name    enumeral_type,
    Pcte_type_name    attribute_type,
    Pcte_natural      *position
);
```

(7)
```
Pcte_error_type Pcte_h_ws_get_enumeral_type_position (
    Pcte_type_reference    enumeral_type,
    Pcte_type_reference    attribute_type,
    Pcte_natural           *position
);
```

```
/* 10.4.4 WS_GET_LINK_TYPE_PROPERTIES */
```

(8)
```
Pcte_error_type Pcte_ws_get_link_type_properties (
    Pcte_type_name              type,
    Pcte_link_type_properties   *properties,
    Pcte_key_types              *key_types,
    Pcte_type_name              reverse
);
```

(9)
```
Pcte_error_type Pcte_h_ws_get_link_type_properties (
    Pcte_type_reference         type,
    Pcte_link_type_properties   *properties,
    Pcte_h_key_types            *key_types,
    Pcte_type_reference         *reverse
);
```

(10) /* If the abstract operation returns no value in *reverse*, **reverse** is returned as a string of zero */
/* length for Pcte_ws_get_link_type_properties and as NULL for */
/* Pcte_h_ws_get_link_type_properties. */

```
/* 10.4.5 WS_GET_OBJECT_TYPE_PROPERTIES */
```

(11)
```
Pcte_error_type Pcte_ws_get_object_type_properties (
    Pcte_type_name      type,
    Pcte_contents_type  *contents_type,
    Pcte_type_names     *parents,
    Pcte_type_names     *children
);
```

(12)
```
Pcte_error_type Pcte_h_ws_get_object_type_properties (
    Pcte_type_reference     type,
    Pcte_contents_type      *contents_type,
    Pcte_type_references    *parents,
    Pcte_type_references    *children
);
```

(13) /* If the abstract operation returns no value in *contents_type* then **contents_type** is set to */
/* PCTE_NO_CONTENTS. */

```
/* 10.4.6 WS_GET_TYPE_KIND */
```

(14)
```
Pcte_error_type Pcte_ws_get_type_kind (
    Pcte_type_name      type,
    Pcte_type_kind      *type_kind
);
```

(15)
```
Pcte_error_type Pcte_h_ws_get_type_kind (
    Pcte_type_reference     type,
    Pcte_type_kind          *type_kind
);
```

```
/* 10.4.7 WS_GET_TYPE_MODES */
```

(16)
```
Pcte_error_type Pcte_ws_get_type_modes (
    Pcte_type_name                  type,
    Pcte_definition_mode_values     *usage_modes
);
```

(17)
```
Pcte_error_type Pcte_h_ws_get_type_modes (
    Pcte_type_reference            type,
    Pcte_definition_mode_values    *usage_modes
);
```

/* 10.4.8 WS_GET_TYPE_NAME */

(18)
```
Pcte_error_type Pcte_ws_get_type_name (
    Pcte_type_name      type,
    Pcte_type_name      name
);
```

(19)
```
Pcte_error_type Pcte_h_ws_get_type_name (
    Pcte_type_reference     type,
    Pcte_type_name          name
);
```

(20) /* If the abstract operation returns no value in *name*, **name** is returned as a string of zero */
/* length. */

/* 10.4.9 WS_SCAN_ATTRIBUTE_TYPE */

(21)
```
Pcte_error_type Pcte_ws_scan_attribute_type (
    Pcte_type_name              type,
    Pcte_attribute_scan_kind    scanning_kind,
    Pcte_type_names             *types
);
```

(22)
```
Pcte_error_type Pcte_h_ws_scan_attribute_type (
    Pcte_type_reference         type,
    Pcte_attribute_scan_kind    scanning_kind,
    Pcte_type_references        *types
);
```

/* 10.4.10 WS_SCAN_ENUMERAL_TYPE */

(23)
```
Pcte_error_type Pcte_ws_scan_enumeral_type (
    Pcte_type_name      type,
    Pcte_type_names     *types
);
```

(24)
```
Pcte_error_type Pcte_h_ws_scan_enumeral_type (
    Pcte_type_reference     type,
    Pcte_type_references    *types
);
```

/* 10.4.11 WS_SCAN_LINK_TYPE */

(25)
```
Pcte_error_type Pcte_ws_scan_link_type (
    Pcte_type_name          type,
    Pcte_link_scan_kind     scanning_kind,
    Pcte_type_names         *types
);
```

(26)
```
Pcte_error_type Pcte_h_ws_scan_link_type (
     Pcte_type_reference      type,
     Pcte_link_scan_kind      scanning_kind,
     Pcte_type_references     *types
);
```

```
/* 10.4.12 WS_SCAN_OBJECT_TYPE */
```

(27)
```
Pcte_error_type Pcte_ws_scan_object_type (
     Pcte_type_name           type,
     Pcte_object_scan_kind    scanning_kind,
     Pcte_type_names          *types
);
```

(28)
```
Pcte_error_type Pcte_h_ws_scan_object_type (
     Pcte_type_reference      type,
     Pcte_object_scan_kind    scanning_kind,
     Pcte_type_references     *types
);
```

```
/* 10.4.13 WS_SCAN_TYPES */
```

(29)
```
Pcte_error_type Pcte_ws_scan_types (
     Pcte_type_kind       kind,
     Pcte_type_names      *types
);
```

(30)
```
Pcte_error_type Pcte_h_ws_scan_types (
     Pcte_type_kind           kind,
     Pcte_type_references     *types
);
```

(31)
```
Pcte_error_type Pcte_ws_scan_all_types (
     Pcte_type_names      *types
);
```

(32)
```
Pcte_error_type Pcte_h_ws_scan_all_types (
     Pcte_type_references     *types
);
```

(33)
```
#endif /* !PCTE_SMS_INCLUDED */
```

11 Volumes, devices, and archives

(1)
```
/* The header <Pcte/devices.h> */
```

(2)
```
#ifndef PCTE_DEVICES_INCLUDED
#define PCTE_DEVICES_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
#include <Pcte/sequences.h>
```

11.1 Volume, device, and archive datatypes

(1)
```
typedef Pcte_natural Pcte_volume_identifier;
```

(2)
```
typedef struct {
    Pcte_natural              total_blocks;
    Pcte_natural              free_blocks;
    Pcte_natural              block_size;
    Pcte_natural              num_objects;
    Pcte_volume_identifier    volume_identifier;
} Pcte_volume_status;
```

(3)
```
typedef Pcte_natural Pcte_device_identifier;
```

(4)
```
typedef enum {
    PCTE_PARTIAL, PCTE_COMPLETE
} Pcte_archive_status;
```

(5)
```
typedef Pcte_natural Pcte_archive_identifier;
```

(6)
```
#include <Pcte/discretionary.h>
```

11.2 Volume, device, and archive operations

```
/* 11.2.1 ARCHIVE_CREATE */
```

(1)
```
Pcte_error_type Pcte_archive_create (
    Pcte_natural                 archive_identifier,
    Pcte_object_reference        on_same_volume_as,
    Pcte_atomic_access_rights    *access_mask,
    Pcte_object_reference        *new_archive
);
```

```
/* 11.2.2 ARCHIVE_REMOVE */
```

(2)
```
Pcte_error_type Pcte_archive_remove (
    Pcte_object_reference        archive
);
```

```
/* 11.2.3 ARCHIVE_RESTORE */
```

(3)
```
Pcte_error_type Pcte_archive_restore (
    Pcte_object_reference        device,
    Pcte_object_reference        archive,
    Pcte_object_references       objects,
    Pcte_object_reference        on_same_volume_as,
    Pcte_archive_status          *restoring_status
);
```

(4)
```
Pcte_error_type Pcte_archive_restore_all (
    Pcte_object_reference        device,
    Pcte_object_reference        archive,
    Pcte_object_reference        on_same_volume_as,
    Pcte_archive_status          *restoring_status
);
```

(5)
```
/*  The effect of specifying scope as a set of object designator to the abstract operation is      */
/*  achieved by the operation Pcte_archive_restore.  The effect of specifying scope as ALL to   */
/*  the abstract operation is achieved by the operation Pcte_archive_restore_all.                 */
```

```
       /* 11.2.4 ARCHIVE_SAVE */
(6)    Pcte_error_type Pcte_archive_save (
           Pcte_object_reference       device,
           Pcte_object_reference       archive,
           Pcte_object_references      objects,
           Pcte_archive_status         *archiving_status
       );
       /* 11.2.5 DEVICE_CREATE */
(7)    Pcte_error_type Pcte_device_create (
           Pcte_object_reference       station,
           Pcte_type_name              device_type,
           Pcte_atomic_access_rights   *access_mask,
           Pcte_natural                device_identifier,
           Pcte_string                 *device_characteristics,
           Pcte_object_reference       *new_device
       );
(8)    Pcte_error_type Pcte_h_device_create (
           Pcte_object_reference       station,
           Pcte_type_reference         device_type,
           Pcte_atomic_access_rights   *access_mask,
           Pcte_natural                device_identifier,
           Pcte_string                 *device_characteristics,
           Pcte_object_reference       *new_device
       );
       /* 11.2.6 DEVICE_REMOVE */
(9)    Pcte_error_type Pcte_device_remove (
           Pcte_object_reference   device
       );
       /* 11.2.7 LINK_GET_DESTINATION_ARCHIVE */
(10)   Pcte_error_type Pcte_link_get_destination_archive (
           Pcte_object_reference   origin,
           Pcte_link_name          link,
           Pcte_archive_identifier *archive_identifier
       );
(11)   Pcte_error_type Pcte_h_link_get_destination_archive (
           Pcte_object_reference   origin,
           Pcte_link_reference     link,
           Pcte_archive_identifier *archive_identifier
       );
```

```
        /* 11.2.8 VOLUME_CREATE */
(12)    Pcte_error_type Pcte_volume_create (
            Pcte_object_reference       device,
            Pcte_natural                volume_id,
            Pcte_atomic_access_rights   *access_mask,
            Pcte_string                 *volume_characteristics,
            Pcte_object_reference       *new_volume
        );
        /* 11.2.9 VOLUME_DELETE */
(13)    Pcte_error_type Pcte_volume_delete (
            Pcte_object_reference   volume
        );
        /* 11.2.10 VOLUME_GET_STATUS */
(14)    Pcte_error_type Pcte_volume_get_status (
            Pcte_object_reference   volume,
            Pcte_volume_status      *volume_status
        );
        /* 11.2.11 VOLUME_MOUNT */
(15)    Pcte_error_type Pcte_volume_mount (
            Pcte_object_reference       device,
            Pcte_volume_identifier      volume_identifier,
            Pcte_boolean                read_only
        );
        /* 11.2.12 VOLUME_UNMOUNT */
(16)    Pcte_error_type Pcte_volume_unmount (
            Pcte_object_reference   volume
        );
(16)    #endif /* !PCTE_DEVICES_INCLUDED */
```

11.3 Clusters

```
(1)     /* The header <Pcte/clusters.h> */
(2)     #ifndef     PCTE_CLUSTERS_INCLUDED
        #define     PCTE_CLUSTERS_INCLUDED      1
(3)     #include  <Pcte/types.h>
        #include  <Pcte/references.h>
        #include  <Pcte/sequences.h>
        #include  <Pcte/security.h>
        /* 11.3.1 CLUSTER_CREATE */
(4)     Pcte_error_type Pcte_cluster_create (
            Pcte_object_reference        volume,
            Pcte_natural                 cluster_id,
            Pcte_atomic_access_rights    *access_mask,
            Pcte_string                  *cluster_characteristics,
            Pcte_object_reference        *new_cluster
        );
        /* 11.3.2 CLUSTER_DELETE */
(5)     Pcte_error_type Pcte_volume_delete (
            Pcte_object_reference    cluster
        );
        /* 11.3.3 CLUSTER_LIST_OBJECTS */
(6)     Pcte_error_type Pcte_cluster_list_objects (
            Pcte_object_reference     cluster,
            Pcte_type_references      types,
            Pcte_object_references    *objects
        );
(7)     #endif
```

12 Files, pipes, and devices

```
(1)     /*The header <Pcte/contents.h> */
(3)     #ifndef PCTE_CONTENTS_INCLUDED
        #define PCTE_CONTENTS_INCLUDED 1
(3)     #include <Pcte/types.h>
        #include <Pcte/references.h>
```

12.1 File, pipe, and device datatypes

```
(1)     typedef void *Pcte_position_handle;
(2)     typedef enum {
            PCTE_READ_WRITE, PCTE_READ_ONLY, PCTE_WRITE_ONLY,
            PCTE_APPEND_ONLY
        } Pcte_contents_access_mode;
(3)     typedef enum {
            PCTE_FROM_BEGINNING, PCTE_FROM_CURRENT, PCTE_FROM_END
        } Pcte_seek_position;
(4)     typedef enum {
            PCTE_AT_BEGINNING, PCTE_AT_POSITION, PCTE_AT_END
        } Pcte_set_position;
(5)     typedef void *Pcte_contents_handle;
```

(6)
```
typedef enum {
    PCTE_SEQUENTIAL, PCTE_DIRECT, PCTE_SEEK
} Pcte_positioning_style;
```

12.2 File, pipe, and device operations

```
/* 12.2.1 CONTENTS_CLOSE */
```

(1)
```
Pcte_error_type Pcte_contents_close (
    Pcte_contents_handle    contents
);
```

```
/* 12.2.2 CONTENTS_GET_HANDLE_FROM_KEY */
```

(2)
```
Pcte_error_type Pcte_contents_get_handle_from_key (
    Pcte_natural            open_object_key,
    Pcte_contents_handle    *contents
);
```

```
/* 12.2.3 CONTENTS_GET_KEY_FROM_HANDLE */
```

(3)
```
Pcte_error_type Pcte_contents_get_key_from_handle (
    Pcte_contents_handle    contents,
    Pcte_natural            *open_object_key
);
```

```
/* 12.2.4 CONTENTS_GET_POSITION */
```

(4)
```
Pcte_error_type Pcte_contents_get_position (
    Pcte_contents_handle    contents,
    Pcte_position_handle    *position
);
```

```
/* 12.2.5 CONTENTS_HANDLE_DUPLICATE */
```

(5)
```
Pcte_error_type Pcte_contents_handle_duplicate (
    Pcte_contents_handle    contents,
    Pcte_boolean            inheritable,
    Pcte_contents_handle    *new_contents
);
```

(6)
```
Pcte_error_type Pcte_contents_handle_duplicate_to_key (
    Pcte_contents_handle    contents,
    Pcte_natural            new_key,
    Pcte_boolean            inheritable,
    Pcte_contents_handle    *new_contents
);
```

(7) /* The effect of not providing the optional parameter *new_key* to the abstract operation is */
/* achieved by the operation Pcte_contents_handle_duplicate. */

```
/* 12.2.6 CONTENTS_OPEN */
```

(8)
```
Pcte_error_type Pcte_contents_open (
    Pcte_object_reference         object,
    Pcte_contents_access_mode     opening_mode,
    Pcte_boolean                  non_blocking_io,
    Pcte_boolean                  inheritable,
    Pcte_contents_handle          *contents
);
```

```
/* 12.2.7 CONTENTS_READ */
```

(9)
```
Pcte_error_type Pcte_contents_read (
    Pcte_contents_handle    contents,
    Pcte_natural            size,
    Pcte_octet              *data,
    Pcte_natural            *data_size
);
```

(10) /* In *data* the read octets are returned and in *data_size* the number of read octets is returned. */
/* If there is not enough space in *data*, the error PCTE_STRING_TOO_SHORT is raised. */

```
/* 12.2.8 CONTENTS_SEEK */
```

(11)
```
Pcte_error_type Pcte_contents_seek (
    Pcte_contents_handle    contents,
    Pcte_integer            offset,
    Pcte_seek_position      whence,
    Pcte_natural            *new_position
);
```

```
/* 12.2.9 CONTENTS_SET_POSITION */
```

(12)
```
Pcte_error_type Pcte_contents_set_position (
    Pcte_contents_handle    contents,
    Pcte_position_handle    position_handle,
    Pcte_set_position       set_mode
);
```

```
/* 12.2.10 CONTENTS_SET_PROPERTIES */
```

(13)
```
Pcte_error_type Pcte_contents_set_properties (
    Pcte_contents_handle    contents,
    Pcte_positioning_style  positioning
);
```

```
/* 12.2.11 CONTENTS_TRUNCATE */
```

(14)
```
Pcte_error_type Pcte_contents_truncate (
    Pcte_contents_handle    contents
);
```

```
      /* 12.2.12 CONTENTS_WRITE */
(15)  Pcte_error_type Pcte_contents_write (
          Pcte_contents_handle    contents,
          Pcte_octet              *data,
          Pcte_natural            data_size,
          Pcte_natural            *actual_size
      );
(16)  /*  In data the octets to be written have to be provided and in data_size the number of octets to */
      /*  be written has to be provided.  If data_size is bigger than the number ofoctets allocated in  */
      /*  data, the error PCTE_ACCESS_AT_INVALID_ADDRESS is.raised.                                     */
      /* 12.2.13 DEVICE_GET_CONTROL */
(17)  Pcte_error_type Pcte_device_get_control (
          Pcte_contents_handle    contents,
          Pcte_natural            operation,
          Pcte_string             *control_data
      );
      /* 12.2.14 DEVICE_SET_CONTROL */
(18)  Pcte_error_type Pcte_device_set_control (
          Pcte_contents_handle    contents,
          Pcte_natural            operation,
          Pcte_string             *control_data
      );
(19)  Pcte_error_type Pcte_position_handle_discard (
          Pcte_position_handle    *position_handle
      );
(20)  /*  Pcte_position_handle_discard discards position handles created by                            */
      /*  Pcte_contents_get_position.                                                                  */
(21)  /*  Pcte_contents_handle does not correspond to an ECMA-149 contents handle but rather           */
      /*  references to it.                                                                            */
(22)  #endif /* !PCTE_CONTENTS_INCLUDED */
```

13 Process execution

```
(1)   /* The header <Pcte/execution.h> */
(2)   #ifndef PCTE_EXECUTION_INCLUDED
      #define PCTE_EXECUTION_INCLUDED 1
(3)   #include <Pcte/types.h>
      #include <Pcte/references.h>
      #include <Pcte/sequences.h>
      #include <Pcte/discretionary.h>
```

13.1 Process execution datatypes

(1)
```
typedef <implementation-defined> Pcte_address;
```

(2) /* **Pcte_address** corresponds to the PCTE datatype Address which must be defined for each */
/* implementation. */

(3)
```
typedef enum {
    PCTE_SUSPENDED, PCTE_RUNNING,  PCTE_STOPPED
} Pcte_initial_status;
```

(4)
```
#define PCTE_EXIT_SUCCESS            <implementation-defined>
#define PCTE_EXIT_ERROR              <implementation-defined>
#define PCTE_SYSTEM_FAILURE          <implementation-defined>
#define PCTE_ACTIVITY_ABORTED        <implementation-defined>
#define PCTE_UNAVAILABLE             <implementation-defined>
#define PCTE_NULL_TERMINATION        <implementation-defined>
```

(5) /* An implementation may provide further values for the termination status of a process by */
/* extending this list of values. */

(6)
```
typedef void *Pcte_profile_handle;
```

(7)
```
#include <Pcte/mandatory.h>
```

13.2 Process execution operations

```
/* 13.2.1 PROCESS_CREATE */
```

(1)
```
Pcte_error_type Pcte_process_create (
    Pcte_object_reference        static_context,
    Pcte_type_name               process_type,
    Pcte_object_reference        parent,
    Pcte_object_reference        site,
    Pcte_boolean                 implicit_deletion,
    Pcte_atomic_access_rights    *access_mask,
    Pcte_object_reference        *new_process
);
```

(2)
```
Pcte_error_type Pcte_h_process_create (
    Pcte_object_reference        static_context,
    Pcte_type_reference          process_type,
    Pcte_object_reference        parent,
    Pcte_object_reference        site,
    Pcte_boolean                 implicit_deletion,
    Pcte_atomic_access_rights    *access_mask,
    Pcte_object_reference        *new_process
);
```

(3) /* The effect of not providing the optional parameter *parent* to the abstract operation is */
/* achieved by specifying **parent** as Pcte_null_object_reference. The effect of not providing */
/* the optional parameter *site* to the abstract operation is achieved by specifying **site** as */
/* Pcte_null_object_reference. */

/* 13.2.2 PROCESS_CREATE_AND_START */

(4)
```
Pcte_error_type Pcte_process_create_and_start (
    Pcte_object_reference       static_context,
    Pcte_string                 *arguments,
    Pcte_string                 *environment,
    Pcte_object_reference       site,
    Pcte_boolean                implicit_deletion,
    Pcte_atomic_access_rights   *access_mask,
    Pcte_object_reference       *new_process
);
```

(5) /* The effect of not providing the optional parameter *site* to the abstract operation is achieved */
/* by specifying **site** as Pcte_null_object_reference. */

/* 13.2.3 PROCESS_GET_WORKING_SCHEMA */

(6)
```
Pcte_error_type Pcte_process_get_working_schema (
    Pcte_object_reference   process,
    Pcte_name_sequence      *sds_sequence
);
```

(7) /* The effect of not providing the optional parameter *process* to the abstract operation is */
/* achieved by specifying **process** as Pcte_null_object_reference. */

/* 13.2.4 PROCESS_INTERRUPT_OPERATION */

(8)
```
Pcte_error_type Pcte_process_interrupt_operation (
    Pcte_object_reference   process
);
```

/* 13.2.5 PROCESS_RESUME */

(9)
```
Pcte_error_type Pcte_process_resume (
    Pcte_object_reference   process
);
```

/* 13.2.6 PROCESS_SET_ALARM */

(10)
```
Pcte_error_type Pcte_process_set_alarm (
    Pcte_natural   duration
);
```

/* 13.2.7 PROCESS_SET_FILE_SIZE_LIMIT */

(11)
```
Pcte_error_type Pcte_process_set_file_size_limit (
    Pcte_object_reference   process,
    Pcte_natural            fslimit
);
```

(12) /* The effect of not providing the optional parameter *process* to the abstract operation is */
/* achieved by specifying **process** as Pcte_null_object_reference. */

/* 13.2.8 PROCESS_SET_OPERATION_TIME_OUT */

(13)
```
Pcte_error_type Pcte_process_set_operation_time_out (
    Pcte_natural   duration
);
```

```
/* 13.2.9 PROCESS_SET_PRIORITY */
```

(14)
```
Pcte_error_type Pcte_process_set_priority (
    Pcte_object_reference   process,
    Pcte_natural            priority
);
```

(15)
```
/*  The effect of not providing the optional parameter process to the abstract operation is    */
/*  achieved by specifying process as Pcte_null_object_reference.                              */
```

```
/* 13.2.10 PROCESS_SET_REFERENCED_OBJECT */
```

(16)
```
Pcte_error_type Pcte_process_set_referenced_object (
    Pcte_object_reference   process,
    Pcte_key                reference_name,
    Pcte_object_reference   object
);
```

(17)
```
/*  The effect of not providing the optional parameter process to the abstract operation is    */
/*  achieved by specifying process as Pcte_null_object_reference.                              */
```

```
/* 13.2.11 PROCESS_SET_TERMINATION_STATUS */
```

(18)
```
Pcte_error_type Pcte_process_set_termination_status (
    Pcte_integer   termination_status
);
```

```
/* 13.2.12 PROCESS_SET_WORKING_SCHEMA */
```

(19)
```
Pcte_error_type Pcte_process_set_working_schema (
    Pcte_object_reference   process,
    Pcte_name_sequence      sds_sequence
);
```

(20)
```
/*  The effect of not providing the optional parameter process to the abstract operation is    */
/*  achieved by specifying process as Pcte_null_object_reference.                              */
```

```
/* 13.2.13 PROCESS_START */
```

(21)
```
Pcte_error_type Pcte_process_start (
    Pcte_object_reference   process,
    Pcte_string             *arguments,
    Pcte_string             *environment,
    Pcte_object_reference   site,
    Pcte_initial_status     initial_status
);
```

(22)
```
/*  The effect of not providing the optional parameter site to the abstract operation is achieved */
/*  by specifying site as Pcte_null_object_reference.                                             */
```

```
/* 13.2.14 PROCESS_SUSPEND */
```

(23)
```
Pcte_error_type Pcte_process_suspend (
    Pcte_object_reference   process,
    Pcte_natural            alarm
);
```

(24)
```
Pcte_error_type Pcte_process_suspend_unlimited (
    Pcte_object_reference    process
);
```

(25)
/* The effect of not providing the optional parameter *process* to the abstract operation is */
/* achieved by specifying **process** as Pcte_null_object_reference. The effect of not */
/* providing the optional parameter *alarm* is achieved by the operation */
/* Pcte_process_suspend_unlimited. */

/* 13.2.15 PROCESS_TERMINATE */

(26)
```
Pcte_error_type Pcte_process_terminate (
    Pcte_object_reference    process,
    Pcte_integer             termination_status
);
```

(27)
/* The effect of not providing the optional parameter *process* to the abstract operation is */
/* achieved by specifying **process** as Pcte_null_object_reference. The effect of not */
/* providing the optional parameter *termination_status* to the abstract operation is achieved */
/* by specifying **termination_status** as PCTE_NULL_TERMINATION. */

/* 13.2.16 PROCESS_UNSET_REFERENCED_OBJECT */

(28)
```
Pcte_error_type Pcte_process_unset_referenced_object (
    Pcte_object_reference    process,
    Pcte_key                 reference_name
);
```

(29)
/* The effect of not providing the optional parameter *process* to the abstract operation is */
/* achieved by specifying **process** as Pcte_null_object_reference. */

/* 13.2.17 PROCESS_WAIT_FOR_ANY_CHILD */

(30)
```
Pcte_error_type Pcte_process_wait_for_any_child (
    Pcte_integer   *termination_status,
    Pcte_natural   *child
);
```

/* 13.2.18 PROCESS_WAIT_FOR_CHILD */

(31)
```
Pcte_error_type Pcte_process_wait_for_child (
    Pcte_object_reference    child,
    Pcte_integer             *termination_status
);
```

13.3 Security operations

/* 13.3.1 PROCESS_ADOPT_USER_GROUP */

(1)
```
Pcte_error_type Pcte_process_adopt_user_group (
    Pcte_object_reference    process,
    Pcte_object_reference    user_group
);
```

(2)
/* The effect of not providing the optional parameter *process* to the abstract operation is */
/* achieved by specifying **process** as Pcte_null_object_reference. */

```
/* 13.3.2 PROCESS_GET_DEFAULT_ACL */
```

(3)
```
Pcte_error_type Pcte_process_get_default_acl (
    Pcte_acl    *acl
);
```

```
/* 13.3.3 PROCESS_GET_DEFAULT_OWNER */
```

(4)
```
Pcte_error_type Pcte_process_get_default_owner (
    Pcte_group_identifier    *group
);
```

```
/* 13.3.4 PROCESS_SET_ADOPTABLE_FOR_CHILD */
```

(5)
```
Pcte_error_type Pcte_process_set_adoptable_for_child (
    Pcte_object_reference    process,
    Pcte_object_reference    user_group,
    Pcte_boolean             adoptability
);
```

(6) /* The effect of not providing the optional parameter *process* to the abstract operation is */
/* achieved by specifying **process** as Pcte_null_object_reference. */

```
/* 13.3.5 PROCESS_SET_DEFAULT_ACL_ENTRY */
```

(7)
```
Pcte_error_type Pcte_process_set_default_acl_entry (
    Pcte_object_reference           process,
    Pcte_group_identifier           group,
    Pcte_requested_access_rights    *modes
);
```

(8) /* The effect of not providing the optional parameter *process* to the abstract operation is */
/* achieved by specifying **process** as Pcte_null_object_reference. */

```
/* 13.3.6 PROCESS_SET_DEFAULT_OWNER */
```

(9)
```
Pcte_error_type Pcte_process_set_default_owner (
    Pcte_object_reference    process,
    Pcte_group_identifier    group
);
```

(10) /* The effect of not providing the optional parameter process to the abstract operation is */
/* achieved by specifying process as Pcte_null_object_reference. */

```
/* 13.3.7 PROCESS_SET_USER */
```

(11)
```
Pcte_error_type Pcte_process_set_user (
    Pcte_object_reference    user,
    Pcte_object_reference    user_group
);
```

13.4 Profiling operations

```
/* 13.4.1 PROCESS_PROFILING_OFF */
```

(1)
```
Pcte_error_type Pcte_process_profiling_off (
    Pcte_profile_handle  handle,
    Pcte_buffer          *buffer
);
```

```
/* 13.4.2 PROCESS_PROFILING_ON */
```

(2)
```
Pcte_error_type Pcte_process_profiling_on (
    Pcte_address         start,
    Pcte_address         end,
    Pcte_natural         count,
    Pcte_profile_handle  *handle
);
```

13.5 Monitoring operations

```
/* 13.5.1 PROCESS_ADD_BREAKPOINT */
```

(1)
```
Pcte_error_type Pcte_process_add_breakpoint (
    Pcte_object_reference   process,
    Pcte_address            breakpoint
);
```

```
/* 13.5.2 PROCESS_CONTINUE */
```

(2)
```
Pcte_error_type Pcte_process_continue (
    Pcte_object_reference   process
);
```

```
/* 13.5.3 PROCESS_PEEK */
```

(3)
```
Pcte_error_type Pcte_process_peek (
    Pcte_object_reference   process,
    Pcte_address            address,
    Pcte_octet              *process_data,
    Pcte_natural            *process_data_size
);
```

(4)
/* In **process_data_size** the number of octets which are to be read has to be provided. In */
/* **process_data** the read octets are returned and in **process_data_size** the number of read */
/* octets is returned. If there is not enough space in **process_data**, the error */
/* PCTE_STRING_TOO_SHORT is raised. */

```
/* 13.5.4 PROCESS_POKE */
```

(5)
```
Pcte_error_type Pcte_process_poke (
    Pcte_object_reference   process,
    Pcte_address            address,
    Pcte_octet              *process_data,
    Pcte_natural            process_data_size
);
```

(6)
```
/*  In process_data the octets to be written have to be provided and in process_data_size the */
/*  number of octets to be written has to be provided.  If process_data_size is bigger than the */
/*  number of octets allocated in process_data, the error                                        */
/*  PCTE_ACCESS_AT_INVALID_ADDRESS is raised.
```

```
/* 13.5.5 PROCESS_REMOVE_BREAKPOINT */
```

(7)
```
Pcte_error_type Pcte_process_remove_breakpoint (
    Pcte_object_reference    process,
    Pcte_address             breakpoint
);
```

```
/* 13.5.6 PROCESS_WAIT_FOR_BREAKPOINT */
```

(8)
```
Pcte_error_type Pcte_process_wait_for_breakpoint (
    Pcte_object_reference    process,
    Pcte_address             *breakpoint
);
```

(9)
```
/*  If the abstract operation returns no value in breakpoint then breakpoint is set to NULL.      */
```

(10)
```
#endif /* !PCTE_EXECUTION_INCLUDED */
```

14 Message queues

(1)
```
/* The header <Pcte/messages.h> */
```

(2)
```
#ifndef PCTE_MESSAGES_INCLUDED
#define PCTE_MESSAGES_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
#include <Pcte/sequences.h>
```

14.1 Message queue datatypes

(1)
```
typedef enum {
    PCTE_INTERRUPT_MSG, PCTE_QUIT_MSG, PCTE_FINISH_MSG,
    PCTE_SUSPEND_MSG, PCTE_END_MSG, PCTE_ABORT_MSG,
    PCTE_DEADLOCK_MSG, PCTE_WAKE_MSG
} Pcte_standard_message_type;
```

(2)
```
typedef enum {
    PCTE_MODIFICATION_MSG, PCTE_CHANGE_MSG, PCTE_DELETE_MSG,
    PCTE_MOVE_MSG, PCTE_NOT_ACCESSIBLE_MSG, PCTE_LOST_MSG
} Pcte_notification_message_type;
```

(3)
```
typedef struct {
    enum {
        PCTE_STANDARD_MESSAGE, PCTE_NOTIFICATION_MESSAGE,
        PCTE_IMPLEMENTATION_MESSAGE, PCTE_UNDEFINED_MESSAGE
    } message_kind;
    union {
        Pcte_standard_message_type        standard;
        Pcte_notification_message_type    notification;
        Pcte_natural                      implementation;
        Pcte_natural                      undefined;
    } type;
} Pcte_message_type;
```

(4)
```
#define Pcte_all_message_types (Pcte_message_types) NULL
```

(5)
```
typedef struct {
    Pcte_string          data;
    Pcte_message_type    message_type;
} Pcte_message;
```

(6)
```
typedef struct {
    Pcte_message     message;
    Pcte_natural     position;
    Pcte_boolean     message_received;
} Pcte_received_message;
```

(7)
```
typedef void (* Pcte_handler) (Pcte_object_reference message_queue);
```

14.2 Message queue operations

/* 14.2.1 MESSAGE_DELETE */

(1)
```
Pcte_error_type Pcte_message_delete (
    Pcte_object_reference    queue,
    Pcte_natural             position
);
```

/* 14.2.2 MESSAGE_PEEK */

(2)
```
Pcte_error_type Pcte_message_peek (
    Pcte_object_reference    queue,
    Pcte_message_types       type,
    Pcte_natural             position,
    Pcte_received_message    *message
);
```

(3) /* The effect of specifying *types* as ALL_MESSAGE_TYPES to the abstract operation is */
/* achieved by specifying **types** as Pcte_all_message_types. The effect of not providing the */
/* optional parameter *position* to the abstract operation is achieved by specifying **position** */
/* as 0. If no message is returned, message_received is set to PCTE_FALSE. */

```
     /* 14.2.3 MESSAGE_RECEIVE_NO_WAIT */
(4)  Pcte_error_type Pcte_message_receive_no_wait (
         Pcte_object_reference    queue,
         Pcte_message_types       types,
         Pcte_natural             position,
         Pcte_received_message    *message
     );
```

(5) /* The effect of specifying *types* as ALL_MESSAGE_TYPES to the abstract operation is */
/* achieved by specifying **types** as Pcte_all_message_types. The effect of not providing the */
/* optional parameter *position* to the abstract operation is achieved by specifying **position** */
/* as 0. If no message is returned, message_received is set to PCTE_FALSE. */

```
     /* 14.2.4 MESSAGE_RECEIVE_WAIT */
(6)  Pcte_error_type Pcte_message_receive_wait (
         Pcte_object_reference    queue,
         Pcte_message_types       types,
         Pcte_natural             position,
         Pcte_received_message    *message
     );
```

(7) /* The effect of not providing the optional parameter *position* to the abstract operation is */
/* is achieved by specifying **position** as 0. */

```
     /* 14.2.5 MESSAGE_SEND_NO_WAIT */
(8)  Pcte_error_type Pcte_message_send_no_wait (
         Pcte_object_reference    queue,
         Pcte_message             *message
     );

     /* 14.2.6 MESSAGE_SEND_WAIT */
(9)  Pcte_error_type Pcte_message_send_wait (
         Pcte_object_reference    queue,
         Pcte_message             *message
     );

     /* 14.2.7 QUEUE_EMPTY */
(10) Pcte_error_type Pcte_queue_empty (
         Pcte_object_reference    queue
     );

     /* 14.2.8 QUEUE_HANDLER_DISABLE */
(11) Pcte_error_type Pcte_queue_handler_disable (
         Pcte_object_reference    queue
     );
```

```
        /* 14.2.9 QUEUE_HANDLER_ENABLE */
(12)    Pcte_error_type Pcte_queue_handler_enable (
            Pcte_object_reference   queue,
            Pcte_message_types      types,
            Pcte_handler            handler
        );
(13)    /*  The effect of specifying types as ALL_MESSAGE_TYPES to the abstract operation is    */
        /*  achieved by specifying types as Pcte_all_message_types.                             */
        /* 14.2.10 QUEUE_RESERVE */
(14)    Pcte_error_type Pcte_queue_reserve (
            Pcte_object_reference   queue
        );
        /* 14.2.11 QUEUE_RESTORE */
(15)    Pcte_error_type Pcte_queue_restore (
            Pcte_object_reference   queue,
            Pcte_object_reference   file
        );
        /* 14.2.12 QUEUE_SAVE */
(16)    Pcte_error_type Pcte_queue_save (
            Pcte_object_reference   queue,
            Pcte_object_reference   file
        );
        /* 14.2.13 QUEUE_SET_TOTAL_SPACE */
(17)    Pcte_error_type Pcte_queue_set_total_space (
            Pcte_object_reference   queue,
            Pcte_natural            total_space
        );
        /* 14.2.14 QUEUE_UNRESERVE */
(18)    Pcte_error_type Pcte_queue_unreserve (
            Pcte_object_reference   queue
        );
(19)    #endif /* !PCTE_MESSAGES_INCLUDED */
```

15 Notification

```
(1)     /* The header <Pcte/notification.h> */
(2)     #ifndef PCTE_NOTIFICATION_INCLUDED
        #define PCTE_NOTIFICATION_INCLUDED 1
(3)     #include <Pcte/types.h>
        #include <Pcte/references.h>
        #include <Pcte/messages.h>
```

15.1 Notification datatypes

(1)
```
typedef enum {
    PCTE_MODIFICATION_EVENT     =  1<<0,
    PCTE_CHANGE_EVENT           =  1<<1,
    PCTE_DELETE_EVENT           =  1<<2,
    PCTE_MOVE_EVENT             =  1<<3
} Pcte_access_event;
```

(2)
```
typedef Pcte_natural Pcte_access_events;
```

15.2 Notification operations

```
/* 15.2.1 NOTIFICATION_MESSAGE_GET_KEY */
```

(1)
```
Pcte_error_type Pcte_notification_message_get_key (
    Pcte_received_message    message,
    Pcte_natural             *notifier_key
);
```

```
/* 15.2.2 NOTIFY_CREATE */
```

(2)
```
Pcte_error_type Pcte_notify_create (
    Pcte_natural             notifier_key,
    Pcte_object_reference    queue,
    Pcte_object_reference    object
);
```

```
/* 15.2.3 NOTIFY_DELETE */
```

(3)
```
Pcte_error_type Pcte_notify_delete (
    Pcte_natural             notifier_key,
    Pcte_object_reference    queue
);
```

```
/* 15.2.4 NOTIFY_SWITCH_EVENTS */
```

(4)
```
Pcte_error_type Pcte_notify_switch_events (
    Pcte_natural             notifier_key,
    Pcte_object_reference    queue,
    Pcte_access_events       access_events
);
```

(5)
```
#endif /* !PCTE_NOTIFICATION_INCLUDED */
```

16 Concurrency and integrity control

(1)
```
/* The header <Pcte/activities.h> */
```

(2)
```
#ifndef PCTE_ACTIVITIES_INCLUDED
#define PCTE_ACTIVITIES_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
#include <Pcte/oms.h>
```

16.1 Concurrency and integrity control datatypes

(1)
```
typedef enum {
    PCTE_UNPROTECTED, PCTE_PROTECTED, PCTE_TRANSACTION
} Pcte_activity_class;
```

(2)
```
typedef enum {
    PCTE_READ_UNPROTECTED, PCTE_READ_SEMIPROTECTED,
    PCTE_WRITE_UNPROTECTED, PCTE_WRITE_SEMIPROTECTED,
    PCTE_DELETE_UNPROTECTED, PCTE_DELETE_SEMIPROTECTED,
    PCTE_READ_PROTECTED, PCTE_WRITE_PROTECTED,
    PCTE_DELETE_PROTECTED, PCTE_WRITE_TRANSACTIONED,
    PCTE_DELETE_TRANSACTIONED, PCTE_READ_DEFAULT,
    PCTE_WRITE_DEFAULT, PCTE_DELETE_DEFAULT
} Pcte_lock_set_mode;
```

(3)
```
typedef Pcte_lock_set_mode Pcte_lock_internal_mode;
```

16.2 Concurrency and integrity control operations

```
/* 16.2.1 ACTIVITY_ABORT */
```

(1)
```
Pcte_error_type Pcte_activity_abort (
);
```

```
/* 16.2.2 ACTIVITY_END */
```

(2)
```
Pcte_error_type Pcte_activity_end (
);
```

```
/* 16.2.3 ACTIVITY_START */
```

(3)
```
Pcte_error_type Pcte_activity_start (
    Pcte_activity_class    activity_class
);
```

```
/* 16.2.4 LOCK_RESET_INTERNAL_MODE */
```

(4)
```
Pcte_error_type Pcte_lock_reset_internal_mode (
    Pcte_object_reference    object
);
```

```
/* 16.2.5 LOCK_SET_INTERNAL_MODE */
```

(5)
```
Pcte_error_type Pcte_lock_set_internal_mode (
    Pcte_object_reference      object,
    Pcte_lock_internal_mode    lock_mode,
    Pcte_boolean               wait_flag
);
```

(6)
```
/*  If the value PCTE_READ_DEFAULT, PCTE_WRITE_DEFAULT,                          */
/*  PCTE_DELETE_DEFAULT, PCTE_DELETE_PROTECTED,                                  */
/*  PCTE_WRITE_TRANSACTIONED, or PCTE_DELETE_TRANSACTIONED is passed */
/*  to lock_mode, the error PCTE_VALUE_IS_OUT_OF_RANGE is raised.        */
/* 16.2.6 LOCK_SET_OBJECT */
```

(7)
```
Pcte_error_type Pcte_lock_set_object (
    Pcte_object_reference   object,
    Pcte_lock_set_mode      lock_mode,
    Pcte_boolean            wait_flag,
    Pcte_object_scope       scope
);
```

```
/* 16.2.7 LOCK_UNSET_OBJECT */
```

(8)
```
Pcte_error_type Pcte_lock_unset_object (
    Pcte_object_reference   object,
    Pcte_object_scope       scope
);
```

(9)
```
#endif /* !PCTE_ACTIVITIES_INCLUDED */
```

17 Replication

(1)
```
/* The header <Pcte/replication.h> */
```

(2)
```
#ifndef PCTE_REPLICATION_INCLUDED
#define PCTE_REPLICATION_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
```

17.1 Replication datatypes

(1)
```
/* None. */
```

17.2 Replication operations

```
/* 17.2.1 REPLICA_SET_ADD_COPY_VOLUME */
```

(1)
```
Pcte_error_type Pcte_replica_set_add_copy_volume (
    Pcte_object_reference   replica_set,
    Pcte_object_reference   copy_volume
);
```

```
/* 17.2.2 REPLICA_SET_CREATE */
```

(2)
```
Pcte_error_type Pcte_replica_set_create (
    Pcte_object_reference   master_volume,
    Pcte_natural            identifier,
    Pcte_object_reference   *replica_set
    );
```

```
/* 17.2.3 REPLICA_SET_REMOVE */
```

(3)
```
Pcte_error_type Pcte_replica_set_remove (
    Pcte_object_reference   replica_set
);
```

```
    /* 17.2.4 REPLICA_SET_REMOVE_COPY_VOLUME */
(4) Pcte_error_type Pcte_replica_set_remove_copy_volume (
        Pcte_object_reference    replica_set,
        Pcte_object_reference    copy_volume
    );
    /* 17.2.5 REPLICATED_OBJECT_CREATE */
(5) Pcte_error_type Pcte_replicated_object_create (
        Pcte_object_reference    replica_set,
        Pcte_object_reference    object
    );
    /* 17.2.6 REPLICATED_OBJECT_DELETE_REPLICA */
(6) Pcte_error_type Pcte_replicated_object_delete_replica (
        Pcte_object_reference    object,
        Pcte_object_reference    copy_volume
    );
    /* 17.2.7 REPLICATED_OBJECT_DUPLICATE */
(7) Pcte_error_type Pcte_replicated_object_duplicate (
        Pcte_object_reference    object,
        Pcte_object_reference    volume,
        Pcte_object_reference    copy_volume
    );
    /* 17.2.8 REPLICATED_OBJECT_REMOVE */
(8) Pcte_error_type Pcte_replicated_object_remove (
        Pcte_object_reference    object
    );
    /* 17.2.9 WORKSTATION_SELECT_REPLICA_SET_VOLUME */
(9) Pcte_error_type Pcte_workstation_select_replica_set_volume (
        Pcte_object_reference    workstation,
        Pcte_object_reference    replica_set,
        Pcte_object_reference    volume
    );
    /* 17.2.10 WORKSTATION_UNSELECT_REPLICA_SET_VOLUME */
(10) Pcte_error_type Pcte_workstation_unselect_replica_set_volume (
        Pcte_object_reference    workstation,
        Pcte_object_reference    replica_set
    );
(11) #endif /* !PCTE_REPLICATION_INCLUDED */
```

18 Network connection

(1)
```
/* The header <Pcte/network.h> */
```

(2)
```
#ifndef PCTE_NETWORK_INCLUDED
#define PCTE_NETWORK_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
#include <Pcte/devices.h>
```

18.1 Network connection datatypes

(1)
```
typedef enum {
    PCTE_ACTIVITY_REMOTE_LOCKS          = 1<<0,
    PCTE_ACTIVITY_LOCAL_LOCKS           = 1<<1,
    PCTE_TRANSACTION_REMOTE_LOCKS       = 1<<2,
    PCTE_TRANSACTION_LOCAL_LOCKS        = 1<<3,
    PCTE_QUEUE_REMOTE                   = 1<<4,
    PCTE_QUEUE_LOCAL                    = 1<<5,
    PCTE_RECEIVE_REMOTE                 = 1<<6,
    PCTE_RECEIVE_LOCAL                  = 1<<7,
    PCTE_CHILD_REMOTE                   = 1<<8,
    PCTE_CHILD_LOCAL                    = 1<<9
} Pcte_work_status_item;
```

(2)
```
typedef Pcte_natural Pcte_work_status;
```

(3)
```
typedef enum {
    PCTE_LOCAL, PCTE_CLIENT, PCTE_CONNECTED, PCTE_AVAILABLE
} Pcte_connection_status;
```

(4)
```
typedef Pcte_connection_status Pcte_requested_connection_status;
```

(5)
```
typedef struct {
    Pcte_string               foreign_device;
    Pcte_volume_identifier    administration_volume;
    Pcte_string               volume_characteristics;
    Pcte_device_identifier    device;
    Pcte_string               device_characteristics;
} Pcte_new_administration_volume;
```

(6)
```
typedef struct {
    Pcte_connection_status    connection;
    Pcte_work_status          work;
} Pcte_workstation_status;
```

(7)
```
#define PCTE_MAX_MACHINE_NAME_SIZE PCTE_MAX_NAME_SIZE
```

(8)
```
typedef Pcte_octet Pcte_machine_name[PCTE_MAX_MACHINE_NAME_SIZE + 1];
```

(9)
```
#define PCTE_MAX_NODE_NAME_SIZE PCTE_MAX_NAME_SIZE
```

(10)
```
typedef Pcte_octet Pcte_node_name[PCTE_MAX_NODE_NAME_SIZE + 1];
```

18.2 Network connection operations

```
/* 18.2.1 WORKSTATION_CONNECT */
```

(1)
```
Pcte_error_type Pcte_workstation_connect (
    Pcte_requested_connection_status       status
);
```

(2)
```
/*  If the value PCTE_AVAILABLE is passed to the parameter status the error          */
/*  PCTE_VALUE_OUT_OF_RANGE is raised.                                               */
```

```
/* 18.2.2 WORKSTATION_CREATE */
```

(3)
```
Pcte_error_type Pcte_workstation_create (
    Pcte_natural                          execution_site_identifier,
    Pcte_new_administration_volume        *administration_volume,
    Pcte_atomic_access_rights             *access_mask,
    Pcte_node_name                        node_name,
    Pcte_machine_name                     machine_name
);
```

(4)
```
Pcte_error_type Pcte_workstation_create_with_existing_admin_volume (
    Pcte_natural                execution_site_identifier,
    Pcte_object_reference       existing_administration_volume,
    Pcte_atomic_access_rights   *access_mask,
    Pcte_node_name              node_name,
    Pcte_machine_name           machine_name
);
```

(5)
```
/*  The effect of specifying administration_volume as a new administration volume to the    */
/*  abstract operation is achieved by the operation                                          */
/*  Pcte_workstation_create_with_existing_admin_volume.  The effect of specifying            */
/*  administration_volume as a volume designator to the abstract operation is achieved by the */
/*  operation Pcte_workstation_create.                                                       */
```

```
/* 18.2.3 WORKSTATION_DELETE */
```

(6)
```
Pcte_error_type Pcte_workstation_delete (
    Pcte_object_reference    station
);
```

```
/* 18.2.4 WORKSTATION_DISCONNECT */
```

(7)
```
Pcte_error_type Pcte_workstation_disconnect (
);
```

```
/* 18.2.5 WORKSTATION_GET_STATUS */
```

(8)
```
Pcte_error_type Pcte_workstation_get_status (
    Pcte_object_reference       station,
    Pcte_workstation_status     *status
);
```

(9)
```
/*  The effect of not providing the optional parameter station to the abstract operation is  */
/*  achieved by specifying station as Pcte_null_object_reference.                            */
/* 18.2.6 WORKSTATION_REDUCE_CONNECTION */
```

(10)
```
Pcte_error_type Pcte_workstation_reduce_connection (
    Pcte_object_reference                station,
    Pcte_requested_connection_status     status,
    Pcte_boolean                         force
);
```

(11)
```
/*  The effect of not providing the optional parameter station to the abstract operation is    */
/*  achieved by specifying station as Pcte_null_object_reference. If the value                 */
/*  PCTE_AVAILABLE is passed to the parameter status the error                                 */
/*  PCTE_VALUE_OUT_OF_RANGE is raised.                                                         */
```

18.3 Foreign system operations

```
/* 18.3.1 CONTENTS_COPY_FROM_FOREIGN_SYSTEM */
```

(1)
```
Pcte_error_type Pcte_contents_copy_from_foreign_system (
    Pcte_object_reference    file,
    Pcte_object_reference    foreign_system,
    Pcte_string              *foreign_name,
    Pcte_string              *foreign_parameters
);
```

(2)
```
/*  The effect of not providing the optional parameter foreign_parameters to the abstract    */
/*  operation is achieved by specifying foreign_parameters as NULL.                          */
```

```
/* 18.3.2 CONTENTS_COPY_TO_FOREIGN_SYSTEM */
```

(3)
```
Pcte_error_type Pcte_contents_copy_to_foreign_system (
    Pcte_object_reference    file,
    Pcte_object_reference    foreign_system,
    Pcte_string              *foreign_name,
    Pcte_string              *foreign_parameters
);
```

(4)
```
/*  The effect of not providing the optional parameter foreign_parameters to the abstract    */
/*  operation is achieved by specifying foreign_parameters as NULL.                          */
```

18.4 Time operations

```
/* 18.4.1 TIME_GET */
```

(1)
```
Pcte_error_type Pcte_time_get (
    Pcte_time     *time
);
```

```
/* 18.4.2 TIME_SET */
```

(2)
```
Pcte_error_type Pcte_time_set (
    Pcte_time     time
);
```

(3)
```
#endif /* !PCTE_NETWORK_INCLUDED */
```

19 Discretionary security

(1)
```
/* The header <Pcte/discretionary.h> */
```

(2)
```
#ifndef PCTE_DISCRETIONARY_INCLUDED
#define PCTE_DISCRETIONARY_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
#include <Pcte/sequences.h>
```

19.1 Discretionary security datatypes

(1)
```
#define PCTE_ALL_USERS (Pcte_natural)                1
```

(2)
```
#define PCTE_SECURITY (Pcte_natural)                 2
```

(3)
```
#define PCTE_AUDIT (Pcte_natural)                    3
```

(4)
```
#define PCTE_EXECUTION (Pcte_natural)                4
```

(5)
```
#define PCTE_REPLICATION (Pcte_natural)              5
```

(6)
```
#define PCTE_CONFIGURATION (Pcte_natural)            6
```

(7)
```
#define PCTE_HISTORY (Pcte_natural)                  7
```

(8)
```
#define PCTE_SCHEMA_UPDATE (Pcte_natural)            8
```

(9)
```
typedef enum {
    PCTE_NAVIGATE                          = 1<<0,
    PCTE_READ_ATTRIBUTES                   = 1<<1,
    PCTE_READ_LINKS                        = 1<<2,
    PCTE_READ_CONTENTS                     = 1<<3,
    PCTE_APPEND_LINKS                      = 1<<4,
    PCTE_APPEND_IMPLICIT                   = 1<<5,
    PCTE_APPEND_CONTENTS                   = 1<<6,
    PCTE_WRITE_IMPLICIT                    = 1<<7,
    PCTE_WRITE_ATTRIBUTES                  = 1<<8,
    PCTE_WRITE_LINKS                       = 1<<9,
    PCTE_WRITE_CONTENTS                    = 1<<10,
    PCTE_DELETE                            = 1<<11,
    PCTE_EXECUTE                           = 1<<12,
    PCTE_EXPLOIT_DEVICE                    = 1<<13,
    PCTE_EXPLOIT_SCHEMA                    = 1<<14,
    PCTE_EXPLOIT_CONSUMER_IDENTITY         = 1<<15,
    PCTE_CONTROL_DISCRETIONARY             = 1<<16,
    PCTE_CONTROL_MANDATORY                 = 1<<17,
    PCTE_CONTROL_OBJECT                    = 1<<18,
    PCTE_OWNER                             = 1<<19,
    PCTE_STABILIZE                         = 1<<20
} Pcte_discretionary_access_mode;
```

(10)
```
typedef Pcte_natural Pcte_discretionary_access_modes;
```

(11)
```
typedef struct {
    Pcte_discretionary_access_modes   denied_rights;
    Pcte_discretionary_access_modes   granted_rights;
} Pcte_access_rights;
```

(12)
```
typedef Pcte_access_rights Pcte_atomic_access_rights;
```

(13)
```
typedef Pcte_access_rights Pcte_requested_access_rights;
```

(14)
```
/*  Pcte_access_rights corresponds to the PCTE datatype Access_rights.  Consider a          */
/*  particular access mode, A.  A is represented by an entry E within the bounded set of    */
/*  access mode.  Let DR indicate "denied rights" and GR indicate "granted rights".         */
/*  The following table shows how the entry E is set or not set to specify any given access */
/*  mode value:                                                                             */
/*                                                                                          */
/*                                        GR              DR                                */
/*                  partially-denied      0               0                                 */
/*                  denied                0               1                                 */
/*                  granted               1               0                                 */
/*                  undefined             1               1                                 */
/*                                                                                          */
/*                                                                                          */
/*  In the same way, Pcte_atomic_access_rights and Pcte_requested_access_rights are         */
/*  defined using the following tables:                                                     */
/*                                                                                          */
/*                            Pcte_atomic_access_rights                                     */
/*                                        GR              DR                                */
/*                  denied                0               1                                 */
/*                  granted               1               0                                 */
/*                  undefined             1               1                                 */
/*                                                                                          */
/*                            Pcte_requested_access_rights                                  */
/*                                        GR              DR                                */
/*                  unchanged             0               0                                 */
/*                  denied                0               1                                 */
/*                  granted               1               0                                 */
/*                  undefined             1               1                                 */
```

(15)
```
typedef Pcte_natural Pcte_group_identifier;
```

(16)
```
typedef struct {
    Pcte_group_identifier    group;
    Pcte_access_rights       access_rights;
} Pcte_acl_entry;
```

(17)
```
typedef Pcte_sequence Pcte_acl;
```

(18)
```
/*  The PCTE datatype Acl which is a map from security group identifier to access rights is  */
/*  mapped to the C datatype Pcte_acl.  Pcte_acl is a sequence indicated by PCTE_ACL with */
/*  the C element datatype Pcte_acl_entry.  The component security_group represents the    */
/*  domain of the map whereas the component access_rights indicates the value of the map.   */
```

(19)
```
#include <Pcte/oms.h>
```

19.2 Discretionary access control operations

```
/* 19.2.1 GROUP_GET_IDENTIFIER */
```

(1)

```
Pcte_error_type Pcte_group_get_identifier (
    Pcte_object_reference    group,
    Pcte_group_identifier    *identifier
);
```

```
/* 19.2.2 OBJECT_CHECK_PERMISSION */
```

(2)

```
Pcte_error_type Pcte_object_check_permission (
    Pcte_object_reference             object,
    Pcte_discretionary_access_modes   modes,
    Pcte_object_scope                 scope,
    Pcte_boolean                      *accessible
);
```

```
/* 19.2.3 OBJECT_GET_ACL */
```

(3)

```
Pcte_error_type Pcte_object_get_acl (
    Pcte_object_reference    object,
    Pcte_object_scope        scope,
    Pcte_acl                 *acl
);
```

```
/* 19.2.4 OBJECT_SET_ACL_ENTRY */
```

(4)

```
Pcte_error_type Pcte_object_set_acl_entry (
    Pcte_object_reference          object,
    Pcte_group_identifier          group,
    Pcte_requested_access_rights   *modes,
    Pcte_object_scope              scope
);
```

19.3 Discretionary security administration operations

```
/* 19.3.1 GROUP_INITIALIZE */
```

(1)

```
Pcte_error_type Pcte_group_initialize (
    Pcte_object_reference    group,
    Pcte_group_identifier    *identifier
);
```

```
/* 19.3.2 GROUP_REMOVE */
```

(2)

```
Pcte_error_type Pcte_group_remove (
    Pcte_object_reference    group
);
```

```
/* 19.3.3 GROUP_RESTORE */
```

(3)

```
Pcte_error_type Pcte_group_restore (
    Pcte_object_reference    group,
    Pcte_group_identifier    identifier
);
```

```
/* 19.3.4 PROGRAM_GROUP_ADD_MEMBER */
```

(4)
```
Pcte_error_type Pcte_program_group_add_member (
    Pcte_object_reference    group,
    Pcte_object_reference    program
);
```

```
/* 19.3.5 PROGRAM_GROUP_ADD_SUBGROUP */
```

(5)
```
Pcte_error_type Pcte_program_group_add_subgroup (
    Pcte_object_reference    group,
    Pcte_object_reference    subgroup
);
```

```
/* 19.3.6 PROGRAM_GROUP_REMOVE_MEMBER */
```

(6)
```
Pcte_error_type Pcte_program_group_remove_member (
    Pcte_object_reference    group,
    Pcte_object_reference    program
);
```

```
/* 19.3.7 PROGRAM_GROUP_REMOVE_SUBGROUP */
```

(7)
```
Pcte_error_type Pcte_program_group_remove_subgroup (
    Pcte_object_reference    group,
    Pcte_object_reference    subgroup
);
```

```
/* 19.3.8 USER_GROUP_ADD_MEMBER */
```

(8)
```
Pcte_error_type Pcte_user_group_add_member (
    Pcte_object_reference    group,
    Pcte_object_reference    user
);
```

```
/* 19.3.9 USER_GROUP_ADD_SUBGROUP */
```

(9)
```
Pcte_error_type Pcte_user_group_add_subgroup (
    Pcte_object_reference    group,
    Pcte_object_reference    subgroup
);
```

```
/* 19.3.10 USER_GROUP_REMOVE_MEMBER */
```

(10)
```
Pcte_error_type Pcte_user_group_remove_member (
    Pcte_object_reference    group,
    Pcte_object_reference    user
);
```

```
/* 19.3.11 USER_GROUP_REMOVE_SUBGROUP */
```

(11)
```
Pcte_error_type Pcte_user_group_remove_subgroup (
    Pcte_object_reference    group,
    Pcte_object_reference    subgroup
);
```

(12)
```
#endif /* !PCTE_DISCRETIONARY_INCLUDED */
```

20 Mandatory security

(1)
```
/* The header <Pcte/mandatory.h> */
```

(2)
```
#ifndef PCTE_MANDATORY_INCLUDED
#define PCTE_MANDATORY_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
```

20.1 Mandatory security datatypes

(1)
```
typedef Pcte_string Pcte_security_label;
```

(2)
```
/*  The PCTE datatype Pcte_security_label_string (see 23.1.3.1 of ECMA-149) is mapped to  */
/*  the C datatype Pcte_security_label.                                                    */
```

(3)
```
typedef enum {
    PCTE_NO_FLOAT, PCTE_FLOAT_IN, PCTE_FLOAT_OUT, PCTE_FLOAT_IN_OUT
} Pcte_floating_level;
```

20.2 Mandatory security operations

```
/* 20.2.1 DEVICE_SET_CONFIDENTIALITY_RANGE */
```

(1)
```
Pcte_error_type Pcte_device_set_confidentiality_range (
    Pcte_object_reference    object,
    Pcte_security_label      *high_label,
    Pcte_security_label      *low_label
);
```

```
/* 20.2.2 DEVICE_SET_INTEGRITY_RANGE */
```

(2)
```
Pcte_error_type Pcte_device_set_integrity_range (
    Pcte_object_reference    object,
    Pcte_security_label      *high_label,
    Pcte_security_label      *low_label
);
```

```
/* 20.2.3 EXECUTION_SITE_SET_CONFIDENTIALITY_RANGE */
```

(3)
```
Pcte_error_type Pcte_execution_site_set_confidentiality_range (
    Pcte_object_reference    execution_site,
    Pcte_security_label      *high_label,
    Pcte_security_label      *low_label
);
```

```
/* 20.2.4 EXECUTION_SITE_SET_INTEGRITY_RANGE */
```

(4)
```
Pcte_error_type Pcte_execution_site_set_integrity_range (
    Pcte_object_reference    execution_site,
    Pcte_security_label      *high_label,
    Pcte_security_label      *low_label
);
```

```
     /* 20.2.5 OBJECT_SET_CONFIDENTIALITY_LABEL */

(5)  Pcte_error_type Pcte_object_set_confidentiality_label (
         Pcte_object_reference    object,
         Pcte_security_label      *label
     );

     /* 20.2.6 OBJECT_SET_INTEGRITY_LABEL */

(6)  Pcte_error_type Pcte_object_set_integrity_label (
         Pcte_object_reference    object,
         Pcte_security_label      *label
     );

     /* 20.2.7 VOLUME_SET_CONFIDENTIALITY_RANGE */

(7)  Pcte_error_type Pcte_volume_set_confidentiality_range (
         Pcte_object_reference    volume,
         Pcte_security_label      *high_label,
         Pcte_security_label      *low_label
     );

     /* 20.2.8 VOLUME_SET_INTEGRITY_RANGE */

(8)  Pcte_error_type Pcte_volume_set_integrity_range (
         Pcte_object_reference    volume,
         Pcte_security_label      *high_label,
         Pcte_security_label      *low_label
     );
```

20.3 Mandatory security administration operations

```
     /* 20.3.1  CONFIDENTIALITY_CLASS_INITIALIZE */

(1)  Pcte_error_type Pcte_confidentiality_class_initialize (
         Pcte_object_reference    object,
         Pcte_name                class_name,
         Pcte_object_reference    to_be_dominated
     );
```

(2) /* The effect of not providing the optional parameter *to_be_dominated* to the abstract */
/* operation is achieved by specifying **to_be_dominated** as Pcte_null_object_reference. */

```
     /* 20.3.2 GROUP_DISABLE_FOR_CONFIDENTIALITY_DOWNGRADE */

(3)  Pcte_error_type Pcte_group_disable_for_confidentiality_downgrade (
         Pcte_object_reference    group,
         Pcte_object_reference    confidentiality_class
     );

     /* 20.3.3 GROUP_DISABLE_FOR_INTEGRITY_UPGRADE */

(4)  Pcte_error_type Pcte_group_disable_for_integrity_upgrade (
         Pcte_object_reference    group,
         Pcte_object_reference    integrity_class
     );
```

```
      /* 20.3.4 GROUP_ENABLE_FOR_CONFIDENTIALITY_DOWNGRADE */

(5)   Pcte_error_type Pcte_group_enable_for_confidentiality_downgrade (
          Pcte_object_reference    group,
          Pcte_object_reference    confidentiality_class
      );

      /* 20.3.5 GROUP_ENABLE_FOR_INTEGRITY_UPGRADE */

(6)   Pcte_error_type Pcte_group_enable_for_integrity_upgrade (
          Pcte_object_reference    group,
          Pcte_object_reference    integrity_class
      );

      /* 20.3.6 INTEGRITY_CLASS_INITIALIZE */

(7)   Pcte_error_type Pcte_integrity_class_initialize (
          Pcte_object_reference    object,
          Pcte_name                class_name,
          Pcte_object_reference    to_be_dominated
      );

(8)   /*  The effect of not providing the optional parameter to_be_dominated to the abstract      */
      /*  operation is achieved by specifying to_be_dominated as Pcte_null_object_reference.     */

      /* 20.3.7 USER_EXTEND_CONFIDENTIALITY_CLEARANCE */

(9)   Pcte_error_type Pcte_user_extend_confidentiality_clearance (
          Pcte_object_reference    user,
          Pcte_object_reference    confidentiality_class
      );

      /* 20.3.8 USER_EXTEND_INTEGRITY_CLEARANCE */

(10)  Pcte_error_type Pcte_user_extend_integrity_clearance (
          Pcte_object_reference    user,
          Pcte_object_reference    integrity_class
      );

      /* 20.3.9 USER_REDUCE_CONFIDENTIALITY_CLEARANCE */

(11)  Pcte_error_type Pcte_user_reduce_confidentiality_clearance (
          Pcte_object_reference    user,
          Pcte_object_reference    confidentiality_class
      );

      /* 20.3.10 USER_REDUCE_INTEGRITY_CLEARANCE */

(12)  Pcte_error_type Pcte_user_reduce_integrity_clearance (
          Pcte_object_reference    user,
          Pcte_object_reference    integrity_class
      );
```

20.4 Mandatory security operations for processes

/* 20.4.1 PROCESS_SET_CONFIDENTIALITY_LABEL */

(1)
```
Pcte_error_type Pcte_process_set_confidentiality_label (
    Pcte_object_reference    process,
    Pcte_security_label      *confidentiality_label
);
```

(2)
```
/*  The effect of not providing the optional parameter process to the abstract operation is    */
/*  achieved by specifying process as Pcte_null_object_reference.                               */
```

/* 20.4.2 PROCESS_SET_FLOATING_CONFIDENTIALITY_LEVEL */

(3)
```
Pcte_error_type Pcte_process_set_floating_confidentiality_level (
    Pcte_object_reference    process,
    Pcte_floating_level      floating_mode
);
```

(4)
```
/*  The effect of not providing the optional parameter process to the abstract operation is    */
/*  achieved by specifying process as Pcte_null_object_reference.                               */
```

/* 20.4.3 PROCESS_SET_FLOATING_INTEGRITY_LEVEL */

(5)
```
Pcte_error_type Pcte_process_set_floating_integrity_level (
    Pcte_object_reference    process,
    Pcte_floating_level      floating_mode
);
```

(6)
```
/*  The effect of not providing the optional parameter process to the abstract operation is    */
/*  achieved by specifying process as Pcte_null_object_reference.                               */
```

/* 20.4.4 PROCESS_SET_INTEGRITY_LABEL */

(7)
```
Pcte_error_type Pcte_process_set_integrity_label (
    Pcte_object_reference    process,
    Pcte_security_label      *integrity_label
);
```

(8)
```
/*  The effect of not providing the optional parameter process to the abstract operation is    */
/*  achieved by specifying process as Pcte_null_object_reference.                               */
```

(9)
```
#endif /* !PCTE_MANDATORY_INCLUDED */
```

21 Auditing

(1) /* The header <Pcte/auditing.h> */

(2)
```
#ifndef PCTE_AUDITING_INCLUDED
#define PCTE_AUDITING_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
#include <Pcte/sequences.h>
#include <Pcte/discretionary.h>
#include <Pcte/mandatory.h>
```

21.1 Auditing datatypes

(1)
```
typedef enum {
    PCTE_WRITE, PCTE_READ, PCTE_COPY, PCTE_ACCESS_CONTENTS,
    PCTE_EXPLOIT, PCTE_CHANGE_ACCESS_CONTROL_LIST,
    PCTE_CHANGE_LABEL, PCTE_USE_PREDEFINED_GROUP,
    PCTE_SET_USER_IDENTITY, PCTE_WRITE_CONFIDENTIALITY_VIOLATION,
    PCTE_READ_CONFIDENTIALITY_VIOLATION,
    PCTE_WRITE_INTEGRITY_VIOLATION,
    PCTE_READ_INTEGRITY_VIOLATION, PCTE_COVERT_CHANNEL,
    PCTE_INFORMATION_EVENT
} Pcte_selectable_event_type;
```

(2)
```
typedef enum {
    PCTE_CHANGE_IDENTIFICATION, PCTE_SELECT_AUDIT_EVENT,
    PCTE_SECURITY_ADMINISTRATION
} Pcte_mandatory_event_type;
```

(3)
```
typedef struct {
    enum {
        PCTE_SELECTABLE, PCTE_MANDATORY
    } event_kind;
    union {
        Pcte_selectable_event_type  selectable_event_type;
        Pcte_mandatory_event_type mandatory_event_type;
    } event_type;
} Pcte_event_type;
```

(4) /* **Pcte_event_type** corresponds to the PCTE datatypes Selectable_event_type and */
/* Mandatory_event_type. */

(5)
```
typedef enum {
    PCTE_FAILURE, PCTE_SUCCESS, PCTE_ANY_CODE
} Pcte_selected_return_code;
```

(6)
```
typedef Pcte_selected_return_code Pcte_return_code;
```

(7)
```
typedef struct {
    Pcte_group_identifier    user;
    Pcte_time                time;
    Pcte_exact_identifier    workstation;
    Pcte_event_type          type;
    Pcte_return_code         return_code;
    Pcte_exact_identifier    process;
    Pcte_exact_identifier    object;
} Pcte_object_auditing_record;
```

(8)
```
typedef struct {
    Pcte_group_identifier    user;
    Pcte_time                time;
    Pcte_exact_identifier    workstation;
    Pcte_event_type          type;
    Pcte_return_code         return_code;
    Pcte_exact_identifier    process;
    Pcte_exact_identifier    new_process;
    Pcte_exact_identifier    exploited_object;
} Pcte_exploit_auditing_record;
```

(9)
```
typedef struct {
    Pcte_group_identifier    user;
    Pcte_time                time;
    Pcte_exact_identifier    workstation;
    Pcte_event_type          type;
    Pcte_return_code         return_code;
    Pcte_exact_identifier    process;
    Pcte_string              text;
} Pcte_information_auditing_record;
```

(10)
```
typedef struct {
    Pcte_group_identifier    user;
    Pcte_time                time;
    Pcte_exact_identifier    workstation;
    Pcte_event_type          type;
    Pcte_return_code         return_code;
    Pcte_exact_identifier    process;
    Pcte_exact_identifier    source;
    Pcte_exact_identifier    destination;
} Pcte_copy_auditing_record;
```

(11)
```
typedef struct {
    Pcte_group_identifier    user;
    Pcte_time                time;
    Pcte_exact_identifier    workstation;
    Pcte_event_type          type;
    Pcte_return_code         return_code;
    Pcte_exact_identifier    process;
    Pcte_exact_identifier    group;
} Pcte_security_auditing_record;
```

(12)
```
typedef struct {
    enum {
        PCTE_OBJECT_RECORD, PCTE_EXPLOIT_RECORD,
        PCTE_INFORMATION_RECORD, PCTE_COPY_RECORD,
        PCTE_SECURITY_RECORD
    } type;
    union {
        Pcte_object_auditing_record         object;
        Pcte_exploit_auditing_record        exploit;
        Pcte_information_auditing_record    information;
        Pcte_copy_auditing_record           copy;
        Pcte_security_auditing_record       security;
    } record;
} Pcte_auditing_record;
```

(13)
```
typedef enum {
    PCTE_ENABLED, PCTE_DISABLED
} Pcte_audit_status;
```

(14)
```
typedef struct {
    Pcte_selectable_event_type      selectable_event_type;
    Pcte_selected_return_code       return_code;
} Pcte_general_criterion;
```

(15)
```
typedef struct {
    Pcte_selectable_event_type      selectable_event_type;
    Pcte_group_identifier           user;
} Pcte_user_criterion;
```

(16)
```
typedef struct {
    Pcte_selectable_event_type      selectable_event_type;
    Pcte_security_label             security_label;
} Pcte_confidentiality_criterion;
```

(17)
```
typedef Pcte_confidentiality_criterion Pcte_integrity_criterion;
```

(18)
```
typedef struct {
    Pcte_selectable_event_type      selectable_event_type;
    Pcte_object_reference           object;
} Pcte_object_criterion;
```

(19)
```
typedef enum {
    PCTE_GENERAL, PCTE_USER_DEPENDENT,
    PCTE_CONFIDENTIALITY_DEPENDENT,
    PCTE_INTEGRITY_DEPENDENT, PCTE_OBJECT_DEPENDENT
} Pcte_criterion_type;
```

(20)
```
typedef struct {
    Pcte_criterion_type type;
    union {
        Pcte_general_criterion          general;
        Pcte_user_criterion             user;
        Pcte_confidentiality_criterion  confidentiality;
        Pcte_integrity_criterion        integrity;
        Pcte_object_criterion           object;
    } criterion;
} Pcte_selection_criterion;
```

(21)
```
typedef Pcte_selection_criterion Pcte_specific_criterion;
```

(22)
```
typedef struct {
    Pcte_criterion_type type;
    union {
        Pcte_general_criteria           general;
        Pcte_user_criteria              user;
        Pcte_confidentiality_criteria   confidentiality;
        Pcte_integrity_criteria         integrity;
        Pcte_object_criteria            object;
    } criteria;
} Pcte_criteria;
```

21.2 Auditing operations

```
/* 21.2.1 AUDIT_ADD_CRITERION */
```

(1)
```
Pcte_error_type Pcte_audit_add_criterion (
    Pcte_object_reference       station,
    Pcte_selection_criterion    *criterion
);
```

```
/* 21.2.2 AUDIT_FILE_COPY_AND_RESET */
```

(2)
```
Pcte_error_type Pcte_audit_file_copy_and_reset (
    Pcte_object_reference   source,
    Pcte_object_reference   destination
);
```

```
/* 21.2.3 AUDIT_FILE_READ */
```

(3)
```
Pcte_error_type Pcte_audit_file_read (
    Pcte_object_reference   audit_file,
    Pcte_audit_file         *records
);
```

```
/* 21.2.4 AUDIT_GET_CRITERIA */
```

(4)
```
Pcte_error_type Pcte_audit_get_criteria (
    Pcte_object_reference   station,
    Pcte_criterion_type     criterion_type,
    Pcte_criteria           *criteria
);
```

```
      /* 21.2.5 AUDIT_RECORD_WRITE */
(5)   Pcte_error_type Pcte_auditing_record_write (
          Pcte_string    *text
      );
      /* 21.2.6 AUDIT_REMOVE_CRITERION */
(6)   Pcte_error_type Pcte_audit_remove_criterion (
          Pcte_object_reference        station,
          Pcte_specific_criterion      *criterion
      );
(7)   /*  If a value of type general criterion is passed to criterion then the error                  */
      /*  PCTE_VALUE_OUT_OF_RANGE is raised.                                                           */
(8)   Pcte_error_type Pcte_audit_remove_criterion_of_event_type (
          Pcte_object_reference             station,
          Pcte_selectable_event_type        criterion
      );
(9)   /*  The effect specifying criterion as a specific criterion to the abstract operation is achieved */
      /*  by the operation Pcte_audit_remove_criterion. The effect of specifying criterion as a         */
      /*  selectable event type to the abstract operation is achieved by the operation                  */
      /*  Pcte_audit_remove_criterion_of_event_type.                                                    */
      /* 21.2.7 AUDIT_SELECTION_CLEAR */
(10)  Pcte_error_type Pcte_audit_selection_clear (
          Pcte_object_reference    station
      );
      /* 21.2.8 AUDIT_SWITCH_OFF_SELECTION */
(11)  Pcte_error_type Pcte_audit_switch_off_selection (
          Pcte_object_reference    station
      );
      /* 21.2.9 AUDIT_SWITCH_ON_SELECTION */
(12)  Pcte_error_type Pcte_audit_switch_on_selection (
          Pcte_object_reference    station
      );
      /* 21.2.10 AUDITING_GET_STATUS */
(13)  Pcte_error_type Pcte_auditing_get_status (
          Pcte_object_reference    station,
          Pcte_audit_status        *status
      );
(14)  #endif /* !PCTE_AUDITING_INCLUDED */
```

22 Accounting

(1)
```
/* The header <Pcte/accounting.h> */
```

(2)
```
#ifndef PCTE_ACCOUNTING_INCLUDED
#define PCTE_ACCOUNTING_INCLUDED 1
```

(3)
```
#include <Pcte/types.h>
#include <Pcte/references.h>
#include <Pcte/sequences.h>
#include <Pcte/discretionary.h>
```

22.1 Accounting datatypes

(1)
```
typedef Pcte_natural Pcte_consumer_identifier;
```

(2)
```
typedef Pcte_natural Pcte_resource_identifier;
```

(3)
```
typedef enum {
    PCTE_WORKSTATION, PCTE_FILE, PCTE_PIPE, PCTE_DEVICE,
    PCTE_STATIC_CONTEXT, PCTE_SDS, PCTE_MESSAGE_QUEUE,
    PCTE_INFORMATION
} Pcte_resource_kind;
```

(4)
```
typedef struct {
    Pcte_group_identifier    security_user;
    Pcte_group_identifier    adopted_user_group;
    Pcte_exact_identifier    consumer_group;
    Pcte_exact_identifier    resource_group;
    Pcte_resource_kind       resource_kind;
    Pcte_time                start_time;
    Pcte_float               duration;
    Pcte_float               cpu_time;
    Pcte_float               sys_time;
} Pcte_workstation_accounting_record;
```

(5)
```
typedef Pcte_workstation_accounting_record
  Pcte_static_context_accounting_record;
```

(6)
```
typedef struct {
    Pcte_group_identifier    security_user;
    Pcte_group_identifier    adopted_user_group;
    Pcte_exact_identifier    consumer_group;
    Pcte_exact_identifier    resource_group;
    Pcte_resource_kind       resource_kind;
    Pcte_time                start_time;
} Pcte_sds_accounting_record;
```

(7)
```
typedef struct {
    Pcte_group_identifier    security_user;
    Pcte_group_identifier    adopted_user_group;
    Pcte_exact_identifier    consumer_group;
    Pcte_exact_identifier    resource_group;
    Pcte_resource_kind       resource_kind;
    Pcte_time                start_time;
    Pcte_float               duration;
    Pcte_natural             read_count;
    Pcte_natural             write_count;
    Pcte_natural             read_size;
    Pcte_natural             write_size;
} Pcte_device_accounting_record;
```

(8)
```
typedef Pcte_device_accounting_record Pcte_file_accounting_record;
```

(9)
```
typedef Pcte_device_accounting_record Pcte_pipe_accounting_record;
```

(10)
```
typedef enum {
    PCTE_SEND, PCTE_RECEIVE, PCTE_RESERVE
} Pcte_operation_kind;
```

(11)
```
typedef struct {
    Pcte_group_identifier    security_user;
    Pcte_group_identifier    adopted_user_group;
    Pcte_exact_identifier    consumer_group;
    Pcte_exact_identifier    resource_group;
    Pcte_resource_kind       resource_kind;
    Pcte_time                start_time;
    Pcte_operation_kind      operation;
    Pcte_natural             message_size;
} Pcte_message_queue_accounting_record;
```

(12)
```
typedef struct {
    Pcte_group_identifier    security_user;
    Pcte_group_identifier    adopted_user_group;
    Pcte_exact_identifier    consumer_group;
    Pcte_exact_identifier    resource_group;
    Pcte_resource_kind       resource_kind;
    Pcte_time                start_time;
    Pcte_string              information;
} Pcte_information_accounting_record;
```

(13)
```
typedef struct {
    Pcte_resource_kind resource_kind;
    union {
        Pcte_workstation_accounting_record        workstation;
        Pcte_static_context_accounting_record     static_context;
        Pcte_sds_accounting_record                sds;
        Pcte_device_accounting_record             device;
        Pcte_file_accounting_record               file;
        Pcte_pipe_accounting_record               pipe;
        Pcte_message_queue_accounting_record      message_queue;
        Pcte_information_accounting_record        information;
    } resource;
} Pcte_accounting_record;
```

22.2 Accounting administration operations

```
/* 22.2.1 ACCOUNTING_LOG_COPY_AND_RESET */
```

(1)
```
Pcte_error_type Pcte_accounting_log_copy_and_reset (
    Pcte_object_reference    source_log,
    Pcte_object_reference    destination_log
);
```

```
/* 22.2.2 ACCOUNTING_LOG_READ */
```

(2)
```
Pcte_error_type Pcte_accounting_log_read (
    Pcte_object_reference    log,
    Pcte_accounting_log      *records
);
```

```
/* 22.2.3 ACCOUNTING_OFF */
```

(3)
```
Pcte_error_type Pcte_accounting_off (
    Pcte_object_reference    station
);
```

```
/* 22.2.4 ACCOUNTING_ON */
```

(4)
```
Pcte_error_type Pcte_accounting_on (
    Pcte_object_reference    log,
    Pcte_object_reference    station
);
```

```
/* 22.2.5 ACCOUNTING_RECORD_WRITE */
```

(5)
```
Pcte_error_type Pcte_accounting_record_write (
    Pcte_object_reference    log,
    Pcte_string              *information
);
```

```
/* 22.2.6 CONSUMER_GROUP_INITIALIZE */
```

(6)
```
Pcte_error_type Pcte_consumer_group_initialize (
    Pcte_object_reference        group,
    Pcte_consumer_identifier     *identifier
);
```

```
      /* 22.2.7 CONSUMER_GROUP_REMOVE */
(7)   Pcte_error_type Pcte_consumer_group_remove (
          Pcte_object_reference    group
      );
      /* 22.2.8 RESOURCE_GROUP_ADD_OBJECT */
(8)   Pcte_error_type Pcte_resource_group_add_object (
          Pcte_object_reference    object,
          Pcte_object_reference    group
      );
      /* 22.2.9 RESOURCE_GROUP_INITIALIZE */
(9)   Pcte_error_type Pcte_resource_group_initialize (
          Pcte_object_reference    group,
          Pcte_resource_identifier *identifier
      );
      /* 22.2.10 RESOURCE_GROUP_REMOVE */
(10)  Pcte_error_type Pcte_resource_group_remove (
          Pcte_object_reference    group
      );
      /* 22.2.11 RESOURCE_GROUP_REMOVE_OBJECT */
(11)  Pcte_error_type Pcte_resource_group_remove_object (
          Pcte_object_reference    object,
          Pcte_object_reference    group
      );
```

22.3 Consumer identity operations

```
      /* 22.3.1 PROCESS_SET_CONSUMER_IDENTITY */
(1)   Pcte_error_type Pcte_process_set_consumer_identity (
          Pcte_object_reference    group
      );
      /* 22.3.2 PROCESS_UNSET_CONSUMER_IDENTITY */
(2)   Pcte_error_type Pcte_process_unset_consumer_identity (
      );
(3)   #endif /* !PCTE_ACCOUNTING_INCLUDED */
```

23 References

```
(1)   /* The header <Pcte/references.h> */
(2)   #ifndef PCTE_REFERENCES_INCLUDED
      #define PCTE_REFERENCES_INCLUDED 1
(3)   #include <stdio.h>
      #include <Pcte/types.h>
```

23.1 Reference datatypes

(1)
```
typedef void *Pcte_object_reference;
```
(2)
```
#define Pcte_null_object_reference (Pcte_object_reference) NULL
```
(3)
```
typedef void *Pcte_type_reference;
```
(4)
```
#define Pcte_null_type_reference (Pcte_type_reference) NULL
```
(5)
```
typedef Pcte_type_reference Pcte_attribute_reference;
```
(6)
```
typedef void *Pcte_link_reference;
```
(7)
```
#define Pcte_null_link_reference (Pcte_link_reference) NULL
```
(8)
```
typedef enum {
    PCTE_NOW, PCTE_FIRST_USE, PCTE_EVERY_USE
} Pcte_evaluation_point;
```
(9)
```
typedef enum {
    PCTE_INTERNAL, PCTE_EXTERNAL
} Pcte_evaluation_status;
```
(10)
```
typedef enum {
    PCTE_EQUAL_REF, PCTE_UNEQUAL_REF, PCTE_EXTERNAL_REF
} Pcte_reference_equality;
```
(11)
```
#define PCTE_MAX_NAME_SIZE  <implementation-defined>
```
(12)
```
typedef Pcte_octet Pcte_name [PCTE_MAX_NAME_SIZE + 1];
```
(13)
```
#define PCTE_MAX_TYPE_NAME_SIZE  <implementation-defined>
```
(14)
```
typedef Pcte_octet Pcte_type_name [PCTE_MAX_TYPE_NAME_SIZE + 1];
```
(15)
```
typedef Pcte_type_name Pcte_attribute_name;
```
(16)
```
typedef Pcte_type_name Pcte_type_name_in_sds;
```
(17)
```
#define PCTE_MAX_KEY_SIZE  <implementation-defined>
```
(18)
```
typedef Pcte_octet Pcte_key [PCTE_KEY_SIZE + 1];
```
(19)
```
#define PCTE_MAX_LINK_NAME_SIZE <implementation-defined>
```
(20)
```
typedef Pcte_octet Pcte_link_name [PCTE_MAX_LINK_NAME_SIZE + 1];
```
(21)
```
typedef Pcte_octet *Pcte_pathname;
```
(22)
```
typedef Pcte_octet *Pcte_relative_pathname;
```
(23)
```
typedef struct {
    enum {
        PCTE_NATURAL_KEY, PCTE_STRING_KEY
    } type;
    union {
        Pcte_natural   natural;
        Pcte_key   string;
    } value;
} Pcte_key_value;
```

23.2 Object reference operations

(1)
```
Pcte_error_type Pcte_pathname_discard (
    Pcte_pathname    *pathname
);
```

(2)
```
/*  Pcte_pathname_discard is used to discard a pathname obtained by                        */
/*  Pcte_object_reference_get_path.                                                        */
```

```
/* 23.2.1 OBJECT_REFERENCE_COPY */
```

(3)
```
Pcte_error_type Pcte_object_reference_copy (
    Pcte_object_reference    reference,
    Pcte_evaluation_point    point,
    Pcte_object_reference    *new_reference
);
```

```
/* 23.2.2 OBJECT_REFERENCE_GET_EVALUATION_POINT */
```

(4)
```
Pcte_error_type Pcte_object_reference_get_evaluation_point (
    Pcte_object_reference    reference,
    Pcte_evaluation_point    *point
);
```

```
/* 23.2.3 OBJECT_REFERENCE_GET_PATH */
```

(5)
```
Pcte_error_type Pcte_object_reference_get_path (
    Pcte_object_reference    reference,
    Pcte_pathname            *pathname
);
```

(6)
```
/*  As it is not possible to determine the size of the returned pathname before executing this  */
/*  operation, in this case the implementation allocates memory for the returned pathname,      */
/*  which is a native C language string terminated with a NUL character.  The returned          */
/*  pathname can be discarded using Pcte_pathname_discard.                                      */
```

```
/* 23.2.4 OBJECT_REFERENCE_GET_STATUS */
```

(7)
```
Pcte_error_type Pcte_object_reference_get_status (
    Pcte_object_reference    reference,
    Pcte_evaluation_status   *status
);
```

```
/* 23.2.5 OBJECT_REFERENCE_SET_ABSOLUTE */
```

(8)
```
Pcte_error_type Pcte_object_reference_set_absolute (
    Pcte_pathname            pathname,
    Pcte_evaluation_point    point,
    Pcte_object_reference    *new_reference
);
```

/* 23.2.6 OBJECT_REFERENCE_SET_RELATIVE */

(9)
```
Pcte_error_type Pcte_object_reference_set_relative (
    Pcte_object_reference       reference,
    Pcte_relative_pathname      pathname,
    Pcte_evaluation_point       point,
    Pcte_object_reference       *new_reference
);
```

/* 23.2.7 OBJECT_REFERENCE_UNSET */

(10)
```
Pcte_error_type Pcte_object_reference_unset (
    Pcte_object_reference   *reference
);
```

(11) /* A null pointer is returned in *reference*. */

/* 23.2.8 OBJECT_REFERENCES_ARE_EQUAL */

(12)
```
Pcte_error_type Pcte_object_references_are_equal (
    Pcte_object_reference       first_reference,
    Pcte_object_reference       second_reference,
    Pcte_reference_equality     *equal
);
```

23.3 Link reference operations

/* 23.3.1 LINK_REFERENCE_COPY */

(1)
```
Pcte_error_type Pcte_link_reference_copy (
    Pcte_link_reference     link_reference,
    Pcte_evaluation_point   point,
    Pcte_link_reference     *new_link_reference
);
```

/* 23.3.2 LINK_REFERENCE_GET_EVALUATION_POINT */

(2)
```
Pcte_error_type Pcte_link_reference_get_evaluation_point (
    Pcte_link_reference     link_reference,
    Pcte_evaluation_point   *point
);
```

/* 23.3.3 LINK_REFERENCE_GET_KEY */

(3)
```
Pcte_error_type Pcte_link_reference_get_key (
    Pcte_link_reference  link_reference,
    Pcte_key             key
);
```

/* 23.3.4 LINK_REFERENCE_GET_KEY_VALUE */

(4)
```
Pcte_error_type Pcte_link_reference_get_key_value (
    Pcte_link_reference  link_reference,
    Pcte_natural         index,
    Pcte_key_value       *key_value
);
```

(5)
```
/*  If the abstract operations returns a value of type Natural, key_value.type is set to          */
/*  PCTE_NATURAL_KEY and key_value.natural contains the value of that key attribute.  */
/*  Otherwise key_value.type is set to PCTE_STRING_KEY and key_value.string contains  */
/*  the value of that key attribute.                                                   */
```

```
/* 23.3.5 LINK_REFERENCE_GET_NAME */
```

(6)
```
Pcte_error_type Pcte_link_reference_get_name (
    Pcte_link_reference  link_reference,
    Pcte_link_name       link_name
);
```

```
/* 23.3.6 LINK_REFERENCE_GET_STATUS */
```

(7)
```
Pcte_error_type Pcte_link_reference_get_status (
    Pcte_link_reference      link_reference,
    Pcte_evaluation_status   *status
);
```

```
/* 23.3.7 LINK_REFERENCE_GET_TYPE */
```

(8)
```
Pcte_error_type Pcte_link_reference_get_type (
    Pcte_link_reference  link_reference,
    Pcte_type_reference  *type_reference
);
```

```
/* 23.3.8 LINK_REFERENCE_SET */
```

(9)
```
Pcte_error_type Pcte_link_reference_set_from_name (
    Pcte_link_name           link_name,
    Pcte_evaluation_point    point,
    Pcte_link_reference      *new_link_reference
);
```

(10)
```
Pcte_error_type Pcte_link_reference_set_from_type (
    Pcte_type_reference      type,
    Pcte_evaluation_point    point,
    Pcte_link_reference      *new_link_reference
);
```

(11)
```
Pcte_error_type Pcte_link_reference_set (
    Pcte_key                 key,
    Pcte_type_reference      type,
    Pcte_evaluation_point    point,
    Pcte_link_reference      *new_link_reference
);
```

(12)
```
/*  The effect of providing a value of type Link_name to link_name in the abstract operation  */
/*  is achieved by the operation Pcte_link_reference_set_from_name.  The effect of              */
/*  providing a value of type Type_reference to link_name in the abstract operation is          */
/*  achieved by the operation Pcte_link_reference_set_from_type.  The effect of providing       */
/*  a value of type (Key * Type_reference) to link_name in the abstract operation is achieved  */
/*  by the operation Pcte_link_reference_set.                                                   */
```

/* 23.3.9 LINK_REFERENCE_UNSET */

(13)
```
Pcte_error_type Pcte_link_reference_unset (
    Pcte_link_reference  *link_reference
);
```

(14) /* A null pointer is returned in *link_reference*. */

/* 23.3.10 LINK_REFERENCES_ARE_EQUAL */

(15)
```
Pcte_error_type Pcte_link_references_are_equal (
    Pcte_link_reference          first_link_reference,
    Pcte_link_reference          second_link_reference,
    Pcte_reference_equality      *equal
);
```

23.4 Type reference operations

/* 23.4.1 TYPE_REFERENCE_COPY */

(1)
```
Pcte_error_type Pcte_type_reference_copy (
    Pcte_type_reference       type_reference,
    Pcte_evaluation_point     point,
    Pcte_type_reference       *new_type_reference
);
```

/* 23.4.2 TYPE_REFERENCE_GET_EVALUATION_POINT */

(2)
```
Pcte_error_type Pcte_type_reference_get_evaluation_point (
    Pcte_type_reference       type_reference,
    Pcte_evaluation_point     *point
);
```

/* 23.4.3 TYPE_REFERENCE_GET_IDENTIFIER */

(3)
```
Pcte_error_type Pcte_type_reference_get_identifier (
    Pcte_type_reference  type_reference,
    Pcte_type_name       type_identifier
);
```

/* 23.4.4 TYPE_REFERENCE_GET_NAME */

(4)
```
Pcte_error_type Pcte_type_reference_get_name (
    Pcte_object_reference     sds,
    Pcte_type_reference       type_reference,
    Pcte_type_name            type_name
);
```

(5) /* The effect of not providing the optional parameter *sds* to the abstract operation is */
/* achieved by specifying **sds** as Pcte_null_object_reference. */

/* 23.4.5 TYPE_REFERENCE_GET_STATUS */

(6)
```
Pcte_error_type Pcte_type_reference_get_status (
    Pcte_type_reference       type_reference,
    Pcte_evaluation_status    *status
);
```

```
     /* 23.4.6 TYPE_REFERENCE_SET */

(7)  Pcte_error_type Pcte_type_reference_set (
         Pcte_type_name            type_name,
         Pcte_evaluation_point     point,
         Pcte_type_reference       *new_type_reference
     );

     /* 23.4.7 TYPE_REFERENCE_UNSET */

(8)  Pcte_error_type Pcte_type_reference_unset (
         Pcte_type_reference       *type_reference
     );

(9)  /*  A null pointer is returned in type_reference.                              */

     /* 23.4.8 TYPE_REFERENCES_ARE_EQUAL */

(10) Pcte_error_type Pcte_type_references_are_equal (
         Pcte_type_reference          first_type_reference,
         Pcte_type_reference          second_type_reference,
         Pcte_reference_equality      *equal
     );

(11) #endif /* !PCTE_REFERENCES_INCLUDED */
```

24 Limits

```
(1)  /* The header <Pcte/limits.h> */

(2)  #ifndef PCTE_LIMITS_INCLUDED
     #define PCTE_LIMITS_INCLUDED 1

(3)  #include <Pcte/types.h>
```

24.1 Implementation limit datatypes

```
(1)  /*  The implementation limits MAX_NAME_SIZE, MAX_KEY_SIZE, and                  */
     /*  MAX_LINK_NAME_SIZE, which define the maximum size of the corresponding texts */
     /*  Pcte_name, Pcte_key, and Pcte_link_name, are defined in 23.1.  All other     */
     /*  implementation limits are defined in this clause.                            */

(2)  typedef enum {
         PCTE_STANDARD, PCTE_IMPLEMENTATION, PCTE_REMAINING
     } Pcte_limit_category;

(3)  /*  An implementation of this binding must return three sets of those implementation limits */
     /*  which are defined in this clause:                                            */

     /*  -  STANDARD: The value specified in ECMA-149                                 */
     /*                                                                               */
     /*  -  IMPLEMENTATION: The value supported by the implementation                 */
     /*                                                                               */
     /*  -  REMAINING: Where appropriate, the value remaining at the current time (after the */
     /*     usage of some resources).                                                 */
```

(4)
```
typedef enum {
    PCTE_MAX_ACCESS_CONTROL_LIST_LENGTH,
    PCTE_MAX_ACCOUNT_DURATION, PCTE_DELTA_ACCOUNT_DURATION,
    PCTE_MAX_ACCOUNT_INFORMATION_LENGTH,
    PCTE_MAX_ACTIVITIES,
    PCTE_MAX_ACTIVITIES_PER_PROCESS,
    PCTE_MAX_AUDIT_INFORMATION_LENGTH,
    PCTE_MAX_DIGIT_FLOAT_ATTRIBUTE,
    PCTE_MAX_FILE_SIZE,
    PCTE_MAX_FLOAT_ATTRIBUTE, PCTE_MIN_FLOAT_ATTRIBUTE,
    PCTE_MAX_INTEGER_ATTRIBUTE, PCTE_MIN_INTEGER_ATTRIBUTE,
    PCTE_MAX_KEY_SIZE, PCTE_MAX_KEY_VALUE,
    PCTE_MAX_LINK_REFERENCE_SIZE,
    PCTE_MAX_MESSAGE_QUEUE_SPACE,
    PCTE_MAX_MESSAGE_SIZE,
    PCTE_MAX_MOUNTED_VOLUMES,
    PCTE_MAX_NAME_SIZE,
    PCTE_MAX_NATURAL_ATTRIBUTE,
    PCTE_MAX_OPEN_OBJECTS,
    PCTE_MAX_OPEN_OBJECTS_PER_PROCESS,
    PCTE_MAX_PIPE_SIZE,
    PCTE_MAX_PRIORITY_VALUE,
    PCTE_MAX_PROCESSES,
    PCTE_MAX_PROCESSES_PER_USER,
    PCTE_MAX_SDS_IN_WORKING_SCHEMA,
    PCTE_MAX_SECURITY_GROUPS,
    PCTE_MAX_STRING_ATTRIBUTE_SIZE,
    PCTE_MAX_TIME_ATTRIBUTE, PCTE_MIN_TIME_ATTRIBUTE,
    PCTE_SMALLEST_FLOAT_ATTRIBUTE
} Pcte_limit_name;
```

(5)
```
typedef struct {
    enum {
        PCTE_FLOAT_LIMIT, PCTE_INTEGER_LIMIT,
        PCTE_NATURAL_LIMIT, PCTE_TIME_LIMIT
    } type;
    union {
        Pcte_float v_float;
        Pcte_integer   v_integer;
        Pcte_natural   v_natural;
        Pcte_time      v_time;
    } value;
} Pcte_limit_value;
```

24.2 Implementation limit operations

```
/* 24.2.1 LIMIT_GET_VALUE */
```

(1)
```
Pcte_error_type Pcte_limit_get_value (
    Pcte_limit_category      category,
    Pcte_limit_name          name,
    Pcte_limit_value         *value,
    Pcte_boolean             *unlimited
);
```

(2) /* If there is no limit value, PCTE_TRUE is returned in **unlimited**. Otherwise **unlimited** is */
/* set to PCTE_FALSE and the limit value is returned into the value pointed to by **value**. */

(3)
```
#endif /* !PCTE_LIMITS_INCLUDED */
```

25 Error conditions

(1)
```
/* The header <Pcte/errors.h> */
```

(2)
```
#ifndef PCTE_ERRORS_INCLUDED
#define PCTE_ERRORS_INCLUDED 1
```

25.1 Error condition datatypes

(1)
```
typedef enum {
        PCTE_NO_ERROR,
/* Errors defined in ECMA-149, annex C */
        PCTE_ACCESS_CONTROL_WOULD_NOT_BE_GRANTED,
        PCTE_ACCESS_MODE_IS_INCOMPATIBLE,
        PCTE_ACCESS_MODE_IS_NOT_ALLOWED,
        PCTE_ACCOUNTING_LOG_IS_NOT_ACTIVE,
        PCTE_ACTIVITY_IS_OPERATING_ON_A_RESOURCE,
        PCTE_ACTIVITY_STATUS_IS_INVALID,
        PCTE_ACTIVITY_WAS_NOT_STARTED_BY_CALLING_PROCESS,
        PCTE_ARCHIVE_EXISTS,
        PCTE_ARCHIVE_HAS_ARCHIVED_OBJECTS,
        PCTE_ARCHIVE_IS_INVALID_ON_DEVICE,
        PCTE_ARCHIVE_IS_UNKNOWN,
        PCTE_ATOMIC_ACL_IS_INCOMPATIBLE_WITH_OWNER_CHANGE,
        PCTE_ATTRIBUTE_TYPE_IS_NOT_VISIBLE,
        PCTE_ATTRIBUTE_TYPE_OF_LINK_TYPE_IS_NOT_APPLIED,
        PCTE_ATTRIBUTE_TYPE_OF_OBJECT_TYPE_IS_NOT_APPLIED,
        PCTE_AUDIT_FILE_IS_NOT_ACTIVE,
        PCTE_BREAKPOINT_IS_NOT_DEFINED,
        PCTE_CARDINALITY_IS_INVALID,
        PCTE_CATEGORY_IS_BAD,
        PCTE_CLASS_NAME_IS_INVALID,
        PCTE_CONFIDENTIALITY_CONFINEMENT_WOULD_BE_VIOLATED,
        PCTE_CONFIDENTIALITY_CRITERION_IS_NOT_SELECTED,
        PCTE_CONFIDENTIALITY_LABEL_IS_INVALID,
        PCTE_CONFIDENTIALITY_WOULD_BE_VIOLATED,
        PCTE_CONNECTION_IS_DENIED,
        PCTE_CONSUMER_GROUP_IS_IN_USE,
        PCTE_CONSUMER_GROUP_IS_KNOWN,
PCTE_CONSUMER_GROUP_IS_UNKNOWN,
PCTE_CONTENTS_IS_NOT_EMPTY,
PCTE_CONTENTS_IS_NOT_FILE_CONTENTS,
PCTE_CONTENTS_IS_NOT_OPEN,
PCTE_CONTENTS_OPERATION_IS_INVALID,
PCTE_CONTROL_WOULD_NOT_BE_GRANTED,
PCTE_DATA_ARE_NOT_AVAILABLE,
PCTE_DEFAULT_ACL_WOULD_BE_INCONSISTENT_WITH_DEFAULT_OBJECT_OWNER,
PCTE_DEFAULT_ACL_WOULD_BE_INVALID,
PCTE_DEFINITION_MODE_VALUE_WOULD_BE_INVALID,
PCTE_DESTINATION_OBJECT_TYPE_IS_INVALID,
PCTE_DEVICE_CHARACTERISTICS_ARE_INVALID,
PCTE_DEVICE_CONTROL_OPERATION_IS_INVALID,
PCTE_DEVICE_EXISTS,
PCTE_DEVICE_IS_BUSY,
PCTE_DEVICE_IS_IN_USE,
PCTE_DEVICE_IS_UNKNOWN,
PCTE_DEVICE_LIMIT_WOULD_BE_EXCEEDED,
PCTE_DEVICE_SPACE_IS_FULL,
PCTE_DISCRETIONARY_ACCESS_IS_NOT_GRANTED,
PCTE_ENUMERAL_TYPE_IS_INVALID,
PCTE_ENUMERAL_TYPE_IS_NOT_IN_ATTRIBUTE_VALUE_TYPE,
PCTE_ENUMERAL_TYPE_IS_NOT_VISIBLE,
PCTE_ENUMERAL_TYPES_ARE_MULTIPLE,
PCTE_EVALUATION_STATUS_IS_INCONSISTENT_WITH_EVALUATION_POINT,
PCTE_EVENT_TYPE_IS_NOT_SELECTED,
PCTE_EXECUTION_CLASS_HAS_NO_USABLE_EXECUTION_SITES,
PCTE_EXECUTION_SITE_IS_INACCESSIBLE,
PCTE_EXECUTION_SITE_IS_NOT_IN_EXECUTION_CLASS,
PCTE_EXECUTION_SITE_IS_UNKNOWN,
PCTE_EXTERNAL_LINK_IS_BAD,
PCTE_EXTERNAL_LINK_IS_NOT_DUPLICABLE,
PCTE_FOREIGN_DEVICE_IS_INVALID,
PCTE_FOREIGN_EXECUTION_IMAGE_HAS_NO_SITE,
PCTE_FOREIGN_EXECUTION_IMAGE_IS_BEING_EXECUTED,
PCTE_FOREIGN_OBJECT_IS_INACCESSIBLE,
PCTE_FOREIGN_SYSTEM_IS_INACCESSIBLE,
PCTE_FOREIGN_SYSTEM_IS_INVALID,
PCTE_FOREIGN_SYSTEM_IS_UNKNOWN,
PCTE_GROUP_IDENTIFIER_IS_IN_USE,
PCTE_GROUP_IDENTIFIER_IS_INVALID,
PCTE_IMAGE_IS_ALREADY_ASSOCIATED,
PCTE_IMAGE_IS_DUPLICATED,
PCTE_INTEGRITY_CONFINEMENT_WOULD_BE_VIOLATED,
PCTE_INTEGRITY_CRITERION_IS_NOT_SELECTED,
PCTE_INTEGRITY_LABEL_IS_INVALID,
PCTE_INTEGRITY_WOULD_BE_VIOLATED,
PCTE_INTERPRETER_IS_INTERPRETABLE,
PCTE_INTERPRETER_IS_NOT_AVAILABLE,
PCTE_KEY_ATTRIBUTE_TYPE_UNAPPLY_IS_FORBIDDEN,
PCTE_KEY_IS_BAD,
PCTE_KEY_IS_NOT_SYSTEM_KEY,
PCTE_KEY_SYNTAX_IS_WRONG,
PCTE_KEY_TYPE_IS_BAD,
PCTE_KEY_TYPES_ARE_MULTIPLE,
PCTE_KEY_UPDATE_IS_FORBIDDEN,
PCTE_KEY_VALUE_AND_EVALUATION_POINT_ARE_INCONSISTENT,
PCTE_KEY_VALUE_DOES_NOT_EXIST,
PCTE_LABEL_IS_OUTSIDE_RANGE,
PCTE_LABEL_RANGE_IS_BAD,
PCTE_LAN_ERROR_EXISTS,
PCTE_LIMIT_WOULD_BE_EXCEEDED,
PCTE_LINK_DESTINATION_DOES_NOT_EXIST,
PCTE_LINK_DESTINATION_IS_NOT_VISIBLE,
PCTE_LINK_DOES_NOT_EXIST,
PCTE_LINK_EXCLUSIVENESS_WOULD_BE_VIOLATED,
PCTE_LINK_EXISTS,
PCTE_LINK_NAME_IS_TOO_LONG_IN_CURRENT_WORKING_SCHEMA,
PCTE_LINK_NAME_SYNTAX_IS_WRONG,
PCTE_LINK_REFERENCE_IS_NOT_EVALUATED,
PCTE_LINK_REFERENCE_IS_UNSET,
PCTE_LINK_TYPE_CATEGORY_IS_BAD,
PCTE_LINK_TYPE_IS_NOT_APPLIED_TO_OBJECT_TYPE,
PCTE_LINK_TYPE_IS_NOT_VISIBLE,
PCTE_LINK_TYPE_IS_UNKNOWN,
PCTE_LINK_TYPE_PROPERTIES_AND_KEY_TYPES_ARE_INCONSISTENT,
PCTE_LINK_TYPE_PROPERTIES_ARE_INCONSISTENT,
PCTE_LOCK_COULD_NOT_BE_ESTABLISHED,
PCTE_LOCK_INTERNAL_MODE_CANNOT_BE_CHANGED,
PCTE_LOCK_IS_NOT_EXPLICIT,
PCTE_LOCK_MODE_IS_NOT_ALLOWED,
PCTE_LOCK_MODE_IS_TOO_STRONG,
PCTE_LOWER_BOUND_WOULD_BE_VIOLATED,
PCTE_MANDATORY_CLASS_IS_ALREADY_DOMINATED,
PCTE_MANDATORY_CLASS_IS_KNOWN,
PCTE_MANDATORY_CLASS_IS_UNKNOWN,
PCTE_MANDATORY_CLASS_NAME_IS_IN_USE,
PCTE_MAXIMUM_USAGE_MODE_WOULD_BE_EXCEEDED,
PCTE_MEMORY_ADDRESS_IS_OUT_OF_PROCESS,
PCTE_MEMORY_REGION_IS_NOT_IN_PROFILING_SPACE,
PCTE_MESSAGE_IS_NOT_A_NOTIFICATION_MESSAGE,
PCTE_MESSAGE_POSITION_IS_NOT_VALID,
PCTE_MESSAGE_QUEUE_HAS_BEEN_DELETED,
PCTE_MESSAGE_QUEUE_HAS_BEEN_WOKEN,
PCTE_MESSAGE_QUEUE_HAS_NO_HANDLER,
PCTE_MESSAGE_QUEUE_IS_BUSY,
PCTE_MESSAGE_QUEUE_IS_NOT_RESERVED,
PCTE_MESSAGE_QUEUE_IS_RESERVED,
PCTE_MESSAGE_QUEUE_TOTAL_SPACE_WOULD_BE_TOO_SMALL,
PCTE_MESSAGE_QUEUE_WOULD_BE_TOO_BIG,
PCTE_MESSAGE_TYPES_NOT_FOUND_IN_QUEUE,
PCTE_NON_BLOCKING_IO_IS_INVALID,
PCTE_NOTIFIER_KEY_DOES_NOT_EXIST,
PCTE_NOTIFIER_KEY_EXISTS,
PCTE_OBJECT_ARCHIVING_IS_INVALID,
PCTE_OBJECT_CANNOT_BE_STABILIZED,
PCTE_OBJECT_CRITERION_IS_NOT_SELECTED,
PCTE_OBJECT_HAS_COPIES,
PCTE_OBJECT_HAS_EXTERNAL_LINKS_PREVENTING_DELETION,
PCTE_OBJECT_HAS_GROUP_WHICH_IS_ALREADY_OWNER,
PCTE_OBJECT_HAS_INTERNAL_LINKS_PREVENTING_DELETION,
PCTE_OBJECT_HAS_LINKS_PREVENTING_DELETION,
PCTE_OBJECT_IS_A_PROCESS,
PCTE_OBJECT_IS_A_REPLICA_SET,
PCTE_OBJECT_IS_ALREADY_IN_RESOURCE_GROUP,
PCTE_OBJECT_IS_ARCHIVED,
PCTE_OBJECT_IS_IN_USE_FOR_DELETE,
PCTE_OBJECT_IS_IN_USE_FOR_MOVE,
PCTE_OBJECT_IS_INACCESSIBLE,
PCTE_OBJECT_IS_INACCESSIBLY_ARCHIVED,
PCTE_OBJECT_IS_LOCKED,
PCTE_OBJECT_IS_NOT_ACCOUNTABLE_RESOURCE,
PCTE_OBJECT_IS_NOT_ARCHIVED,
PCTE_OBJECT_IS_NOT_IN_RESOURCE_GROUP,
PCTE_OBJECT_IS_NOT_LOCKED,
PCTE_OBJECT_IS_NOT_MASTER_REPLICATED_OBJECT,
PCTE_OBJECT_IS_NOT_MOVABLE,
PCTE_OBJECT_IS_NOT_ON_ADMINISTRATION_VOLUME,
PCTE_OBJECT_IS_NOT_ON_MASTER_VOLUME_OF_REPLICA_SET,
PCTE_OBJECT_IS_NOT_REPLICABLE,
PCTE_OBJECT_IS_NOT_REPLICATED_ON_VOLUME,
PCTE_OBJECT_IS_OF_WRONG_TYPE,
PCTE_OBJECT_IS_OPERATED_ON,
PCTE_OBJECT_IS_PREDEFINED_REPLICATED,
PCTE_OBJECT_IS_REPLICATED,
PCTE_OBJECT_IS_STABLE,
PCTE_OBJECT_LABEL_CANNOT_BE_CHANGED_IN_TRANSACTION,
PCTE_OBJECT_OWNER_CONSTRAINT_WOULD_BE_VIOLATED,
PCTE_OBJECT_OWNER_VALUE_WOULD_BE_INCONSISTENT_WITH_ATOMIC_ACL,
PCTE_OBJECT_REFERENCE_IS_INTERNAL,
PCTE_OBJECT_REFERENCE_IS_INVALID,
PCTE_OBJECT_REFERENCE_IS_UNSET,
PCTE_OBJECT_TYPE_IS_ALREADY_IN_DESTINATION_SET,
PCTE_OBJECT_TYPE_IS_INVALID,
PCTE_OBJECT_TYPE_IS_NOT_IN_DESTINATION_SET,
PCTE_OBJECT_TYPE_IS_NOT_VISIBLE,
PCTE_OBJECT_TYPE_IS_UNKNOWN,
PCTE_OBJECT_TYPE_WOULD_HAVE_NO_PARENT_TYPE,
PCTE_OBJECT_TYPES_MISMATCH,
PCTE_OPEN_KEY_IS_INVALID,
PCTE_OPENING_MODE_IS_INVALID,
PCTE_OPERATION_HAS_TIMED_OUT,
PCTE_OPERATION_IS_INTERRUPTED,
PCTE_OPERATION_IS_NOT_ALLOWED_ON_TYPE,
PCTE_PARENT_BASIC_TYPES_ARE_MULTIPLE,
PCTE_PATHNAME_SYNTAX_IS_WRONG,
PCTE_POSITION_HANDLE_IS_INVALID,
PCTE_POSITION_IS_INVALID,
PCTE_POSITIONING_IS_INVALID,
PCTE_PREFERENCE_DOES_NOT_EXIST,
PCTE_PREFERRED_LINK_KEY_IS_BAD,
PCTE_PREFERRED_LINK_TYPE_IS_UNSET,
PCTE_PRIVILEGE_IS_NOT_GRANTED,
PCTE_PROCESS_CONFIDENTIALITY_IS_NOT_DOMINATED,
PCTE_PROCESS_HAS_NO_UNTERMINATED_CHILD,
PCTE_PROCESS_INTEGRITY_DOES_NOT_DOMINATE,
PCTE_PROCESS_IS_IN_TRANSACTION,
PCTE_PROCESS_IS_INACCESSIBLE,
PCTE_PROCESS_IS_INITIAL_PROCESS,
PCTE_PROCESS_IS_NOT_ANCESTOR,
PCTE_PROCESS_IS_NOT_CHILD,
PCTE_PROCESS_IS_NOT_TERMINABLE_CHILD,
PCTE_PROCESS_IS_NOT_THE_CALLER,
PCTE_PROCESS_IS_THE_CALLER,
PCTE_PROCESS_IS_UNKNOWN,
PCTE_PROCESS_LABELS_WOULD_BE_INCOMPATIBLE,
PCTE_PROCESS_LACKS_REQUIRED_STATUS,
PCTE_PROCESS_TERMINATION_IS_ALREADY_ACKNOWLEDGED,
PCTE_PROFILING_IS_NOT_SWITCHED_ON,
PCTE_PROGRAM_GROUP_IS_NOT_EMPTY,
PCTE_RANGE_IS_OUTSIDE_RANGE,
PCTE_REFERENCE_CANNOT_BE_ALLOCATED,
PCTE_REFERENCE_NAME_IS_INVALID,
PCTE_REFERENCED_OBJECT_IS_NOT_MUTABLE,
PCTE_REFERENCED_OBJECT_IS_UNSET,
PCTE_RELATIONSHIP_TYPE_PROPERTIES_ARE_INCONSISTENT,
PCTE_REPLICA_SET_COPY_IS_NOT_EMPTY,
PCTE_REPLICA_SET_HAS_COPY_VOLUMES,
PCTE_REPLICA_SET_IS_NOT_EMPTY,
PCTE_REPLICA_SET_IS_NOT_KNOWN,
PCTE_REPLICATED_COPY_IS_IN_USE,
PCTE_REPLICATED_COPY_UPDATE_IS_FORBIDDEN,
PCTE_RESOURCE_GROUP_IS_KNOWN,
PCTE_RESOURCE_GROUP_IS_UNKNOWN,
PCTE_REVERSE_KEY_IS_BAD,
PCTE_REVERSE_KEY_IS_NOT_SUPPLIED,
PCTE_REVERSE_KEY_IS_SUPPLIED,
PCTE_REVERSE_LINK_EXISTS,
PCTE_SDS_IS_IN_A_WORKING_SCHEMA,
PCTE_SDS_IS_KNOWN,
PCTE_SDS_IS_NOT_EMPTY_NOR_VERSION,
PCTE_SDS_IS_UNDER_MODIFICATION,
PCTE_SDS_IS_UNKNOWN,
PCTE_SDS_NAME_IS_DUPLICATE,
PCTE_SDS_NAME_IS_INVALID,
PCTE_SDS_WOULD_APPEAR_TWICE_IN_WORKING_SCHEMA,
PCTE_SECURITY_GROUP_ALREADY_HAS_THIS_SUBGROUP,
PCTE_SECURITY_GROUP_IS_ALREADY_ENABLED,
PCTE_SECURITY_GROUP_IS_IN_USE,
PCTE_SECURITY_GROUP_IS_KNOWN,
PCTE_SECURITY_GROUP_IS_NOT_A_SUBGROUP,
PCTE_SECURITY_GROUP_IS_NOT_ADOPTABLE,
PCTE_SECURITY_GROUP_IS_NOT_ENABLED,
PCTE_SECURITY_GROUP_IS_PREDEFINED,
PCTE_SECURITY_GROUP_IS_REQUIRED_BY_OTHER_GROUPS,
PCTE_SECURITY_GROUP_IS_UNKNOWN,
PCTE_SECURITY_GROUP_WOULD_BE_IN_INVALID_GRAPH,
PCTE_SECURITY_POLICY_WOULD_BE_VIOLATED,
PCTE_STATIC_CONTEXT_CONTENTS_CANNOT_BE_EXECUTED,
PCTE_STATIC_CONTEXT_IS_ALREADY_MEMBER,
PCTE_STATIC_CONTEXT_IS_BEING_WRITTEN,
PCTE_STATIC_CONTEXT_IS_IN_USE,
PCTE_STATIC_CONTEXT_IS_NOT_MEMBER,
PCTE_STATIC_CONTEXT_REQUIRES_TOO_MUCH_MEMORY,
PCTE_STATUS_IS_BAD,
PCTE_TIME_CANNOT_BE_CHANGED,
PCTE_TRANSACTION_CANNOT_BE_COMMITTED,
PCTE_TYPE_HAS_DEPENDENCIES,
PCTE_TYPE_HAS_NO_LOCAL_NAME,
PCTE_TYPE_IDENTIFIER_IS_INVALID,
PCTE_TYPE_IDENTIFIER_SYNTAX_IS_WRONG,
PCTE_TYPE_IDENTIFIER_USAGE_IS_INVALID,
PCTE_TYPE_IS_ALREADY_APPLIED,
PCTE_TYPE_IS_ALREADY_KNOWN_IN_SDS,
PCTE_TYPE_IS_NOT_APPLIED,
PCTE_TYPE_IS_NOT_DESCENDANT,
PCTE_TYPE_IS_NOT_VISIBLE,
PCTE_TYPE_IS_OF_WRONG_KIND,
PCTE_TYPE_IS_UNKNOWN,
PCTE_TYPE_IS_UNKNOWN_IN_SDS,
        PCTE_TYPE_IS_UNKNOWN_IN_WORKING_SCHEMA,
        PCTE_TYPE_NAME_IN_SDS_IS_DUPLICATE,
        PCTE_TYPE_NAME_IS_INVALID,
        PCTE_TYPE_OF_OBJECT_IS_INVALID,
        PCTE_TYPE_REFERENCE_IS_INVALID,
        PCTE_TYPE_REFERENCE_IS_UNSET,
        PCTE_UNLOCKING_IN_TRANSACTION_IS_FORBIDDEN,
        PCTE_UPPER_BOUND_WOULD_BE_VIOLATED,
        PCTE_USAGE_MODE_ON_ATTRIBUTE_TYPE_WOULD_BE_VIOLATED,
        PCTE_USAGE_MODE_ON_LINK_TYPE_WOULD_BE_VIOLATED,
        PCTE_USAGE_MODE_ON_OBJECT_TYPE_WOULD_BE_VIOLATED,
        PCTE_USER_CRITERION_IS_NOT_SELECTED,
        PCTE_USER_GROUP_IS_IN_USE,
        PCTE_USER_GROUP_LACKS_ALL_USERS_AS_SUPERGROUP,
        PCTE_USER_GROUP_WOULD_NOT_HAVE_ALL_USERS_AS_SUPERGROUP,
        PCTE_USER_IS_ALREADY_CLEARED_TO_CLASS,
        PCTE_USER_IS_ALREADY_MEMBER,
        PCTE_USER_IS_IN_USE,
        PCTE_USER_IS_NOT_CLEARED,
        PCTE_USER_IS_NOT_CLEARED_TO_CLASS,
        PCTE_USER_IS_NOT_MEMBER,
        PCTE_USER_IS_UNKNOWN,
        PCTE_VALUE_TYPE_IS_INVALID,
        PCTE_VERSION_GRAPH_IS_INVALID,
        PCTE_VERSION_IS_REQUIRED,
        PCTE_VOLUME_CANNOT_BE_MOUNTED_ON_DEVICE,
        PCTE_VOLUME_EXISTS,
        PCTE_VOLUME_HAS_OBJECT_OUTSIDE_RANGE,
        PCTE_VOLUME_HAS_OBJECTS_IN_USE,
        PCTE_VOLUME_HAS_OTHER_LINKS,
        PCTE_VOLUME_HAS_OTHER_OBJECTS,
        PCTE_VOLUME_IDENTIFIER_IS_INVALID,
        PCTE_VOLUME_IS_ADMINISTRATION_VOLUME,
        PCTE_VOLUME_IS_ALREADY_COPY_VOLUME_OF_REPLICA_SET,
        PCTE_VOLUME_IS_ALREADY_MOUNTED,
        PCTE_VOLUME_IS_FULL,
        PCTE_VOLUME_IS_INACCESSIBLE,
        PCTE_VOLUME_IS_MASTER_VOLUME_OF_REPLICA_SET,
        PCTE_VOLUME_IS_NOT_COPY_VOLUME_OF_REPLICA_SET,
        PCTE_VOLUME_IS_NOT_MASTER_OR_COPY_VOLUME_OF_REPLICA_SET,
        PCTE_VOLUME_IS_READ_ONLY,
        PCTE_VOLUME_IS_UNKNOWN,
        PCTE_WORKSTATION_EXISTS,
        PCTE_WORKSTATION_HAS_NO_CHOICE_OF_VOLUME_FOR_REPLICA_SET,
        PCTE_WORKSTATION_IDENTIFIER_IS_INVALID,
        PCTE_WORKSTATION_IS_BUSY,
        PCTE_WORKSTATION_IS_CONNECTED,
        PCTE_WORKSTATION_IS_NOT_CONNECTED,
        PCTE_WORKSTATION_IS_UNKNOWN,
/* C binding specific errors */
        PCTE_ACCESS_MASK_IS_INVALID,
        PCTE_ACCESS_AT_INVALID_ADDRESS,
        PCTE_OUT_OF_MEMORY,
        PCTE_SEQUENCE_INVALID_TYPE,
        PCTE_SEQUENCE_BAD_HANDLE,
        PCTE_SEQUENCE_OUT_OF_DATA,
        PCTE_SEQUENCE_INVALID_INDEX,
        PCTE_STRING_TOO_SHORT,
        PCTE_VALUE_IS_OUT_OF_RANGE,
        PCTE_VALUE_TYPE_IDENTIFIER_DOES_NOT_MATCH,
/* fine-grain object errors */
        PCTE_CLUSTER_EXISTS,
        PCTE_CLUSTER_HAS_OTHER_LINKS,
        PCTE_CLUSTER_IS_UNKNOWN,
        PCTE_OBJECT_CANNOT_BE_CLUSTERED,
        PCTE_OBJECT_IS_FINE_GRAIN,
/* object orientation errors */
        PCTE_NUMBER_OF_PARAMETERS_IS_WRONG,
        PCTE_OPERATION_METHOD_CANNOT_FOUND,
        PCTE_OPERATION_METHOD_CANNOT_BE_ACTIVATED,
        PCTE_TYPE_IS_ALREADY_CONSTRAINED,
        PCTE PCTE_TYPE_OF_PARAMETER_IS_WRONG
} Pcte_error_type;
```

(2)
```
/*  For each error defined in the PCTE Abstract Specification, the C binding defines an     */
/*  enumeration constant which has the same name.  There are additional error which are     */
/*  specific of the C binding:                                                              */
/*  -  PCTE_ACCESS_AT_INVALID_ADDRESS may be raised when the process is                     */
/*     attempting to access some code or data, at an invalid address.  This error can occur in */
/*     the context of all operations, each time an invalid address is provided as value of a */
/*     pointer argument.                                                                    */
/*  -  PCTE_SEQUENCE_xxx may be raised within operations on sequences.                      */
/*  -  PCTE_VALUE_IS_OUT_OF_RANGE may be raised by any operation which has an               */
/*     input enumeration value as or as part of a parameter, if the value is out of the range of */
/*     allowed enumeration values, or by any operation which has a value of a bounded-set as */
/*     or as part of an input parameter, if any of the undefined bits of the representing natural */
/*     value is set.                                                                        */
/*  -  PCTE_OUT_OF_MEMORY may be raised by any operation if a process is running            */
/*     out of memory.                                                                       */
/*  -  PCTE_VALUE_TYPE_IDENTIFIER_DOES_NOT_MATCH may be raised by any                       */
/*     operation which has a value value of type Pcte_value_type as or as part of an input  */
/*     parameter where the component value.value_type_identifier does not match the value  */
/*     type identifier of the specified attribute.                                          */
/*  -  PCTE_ACCESS_MASK_IS_INVALID may be raised by any operation with an                   */
/*     access_mask parameter (which is of type Pcte_atomic_access_rights) if a value is     */
/*     supplied with both denied rights and granted rights set to 0 for one or more         */
/*     discretionary access modes.                                                          */
/*  -  PCTE_STRING_TOO_SHORT may be raised to indicate that the implementation              */
/*     has not provided enough space to hold the returned value.                            */
```

(3) extern Pcte_error_type Pcte_error_number;

25.2 Error condition operations

(1) /* None. */

(2)

```
#endif /* !PCTE_ERRORS_INCLUDED */
```

Annex A
(normative)

The object orientation module

This annex defines the C 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)
```
/* The header <Pcte/methods.h> */
```
(2)
```
#ifndef PCTE_IMPLEMENTATIONS_INCLUDED
#define PCTE_IMPLEMENTATIONS_INCLUDED 1
```
(3)
```
#include <Pcte/types.h>
```
(4)
```
#include <Pcte/references.h>
```
(5)
```
#include <Pcte/sequences.h>
```
(6)
```
#include <Pcte/oms.h>
```

A.1.1 Object-oriented invocation management datatypes

(1)
```
typedef enum {
   PCTE_CONSTRAINED_TO_ATTRIBUTE,
   PCTE_CONSTRAINED_TO_OBJECT,
   PCTE_CONSTRAINED_TO_INTERFACE
} Pcte_parameter_constraint;
```
(2)
```
typedef struct {
   Pcte_parameter_constraint  constraint;
   union {
       Pcte_attribute_value    *p_value;
       Pcte_object_reference   p_object;
       Pcte_object_reference   p_interface;
   } parameter;
} Pcte_parameter_item;
```
(3)
```
typedef Pcte_sequence Pcte_parameter_items;
```
(4)
```
typedef struct {
   Pcte_object_reference   target_object;
   Pcte_type_name          operation_id;
   Pcte_parameter_items    parameters;
   Pcte_object_reference   context;
} Pcte_method_request;
```
(5)
```
typedef Pcte_sequence Pcte_method_requests;
```

```
(6)      typedef enum {
            PCTE_ADOPT_WORKING_SCHEMA          =       1<<0,
            PCTE_ADOPT_ACTIVITY                =       1<<1,
            PCTE_ADOPT_USER                    =       1<<2,
            PCTE_ADOPT_OPEN_OBJECTS            =       1<<3,
            PCTE_ADOPT_REFERENCE_OBJECTS       =       1<<4,
            PCTE_ADOPT_ALL
         } Pcte_context_adoption;

(7)      #define PCTE_ADOPT_ALL (Pcte_natural)
            (PCTE_ADOPT_WORKING_SCHEMA | \
             PCTE_ADOPT_ACTIVITY | \
             PCTE_ADOPT_USER | \
             PCTE_ADOPT_OPEN_OBJECTS | \
             PCTE_ADOPT_REFERENCE_OBJECTS)

(8)      typedef Pcte_sequence Pcte_context_adoptions;

(9)      typedef void *Pcte_method_request_id;

(10)     typedef Pcte_sequence Pcte_method_request_ids;
```

A.1.2 Object-oriented invocation management operations

```
         /* G.2.2.1 PROCESS_ADOPT_CONTEXT */

(1)      Pcte_error_type Pcte_process_adopt_context (
            Pcte_context_adoptions      context_adoptions;
         );

        /* G.2.2.2 REQUEST_INVOKE */

(2)      Pcte_error_type Pcte_request_invoke (
            Pcte_method_request         *request,
            Pcte_context_adoptions      context_adoptions;
            Pcte_method_request_id      *request_id;
         );

         /* G.2.2.3 REQUEST_SEND */

(3)      Pcte_error_type Pcte_request_send (
            Pcte_method_request         *request,
            Pcte_context_adoptions      context_adoptions;
            Pcte_method_request_id      *request_id;
         );

         /* G.2.2.4 REQUEST_SEND_MULTIPLE */

(4)      Pcte_error_type Pcte_request_send_multiple (
            Pcte_method_requests        requests,
            Pcte_context_adoptions      context_adoptions;
            Pcte_method_request_ids     *request_ids;
         );

(5)      #endif
```

A.2 Object-oriented schema management

(1)
```
/* The header <Pcte/interfaces.h> */
```

(2)
```
#ifndef PCTE_INTERFACES_INCLUDED
#define PCTE_INTERFACES_INCLUDED 1
```

(3)
```
#include <Pcte/references.h>
```

(4)
```
#include <Pcte/sequences.h>
```

A.2.1 Object-oriented schema management datatypes

(1)
```
typedef enum {
   PCTE_NO_OPERATION, PCTE_ALL_OPERATIONS
} Pcte_interface_scope;
```

A.2.2 Object-oriented schema management operations

```
/* G.3.2.1 SDS_APPLY_INTERFACE_TYPE */
```

(1)
```
Pcte_error_type Pcte_sds_apply_interface_type (
   Pcte_object_reference      sds,
   Pcte_type_name_in_sds      interface_type,
   Pcte_type_name_in_sds      type
);
```

```
/* G.3.2.2 SDS_APPLY_OPERATION_TYPE */
```

(2)
```
Pcte_error_type Pcte_sds_apply_operation_type (
   Pcte_object_reference      sds,
   Pcte_type_name_in_sds      operation_type,
   Pcte_type_name_in_sds      type
);
```

```
/* G.3.2.3 SDS_CREATE_DATA_PARAMETER_TYPE */
```

(3)
```
Pcte_error_type Pcte_sds_create_data_parameter_type (
   Pcte_object_reference   sds,
   Pcte_name               local_name,
   Pcte_type_name          data_type,
   Pcte_type_name          new_parameter
);
```

(4)
/*   The effect of not providing the optional parameter *local_name* to the abstract operation   */
/*   is achieved by specifying **local_name** as NULL.   */

```
/* G.3.2.4 SDS_CREATE_INTERFACE_PARAMETER_TYPE */
```

(5)
```
Pcte_error_type Pcte_sds_create_interface_parameter_type (
   Pcte_object_reference   sds,
   Pcte_name               local_name,
   Pcte_type_name          interface_type,
   Pcte_type_name          new_parameter
);
```

(6) /* The effect of not providing the optional parameter *local_name* to the abstract operation */
/* is achieved by specifying **local_name** as NULL. */

/* G.3.2.5 SDS_CREATE_INTERFACE_TYPE */

(7)
```
Pcte_error_type Pcte_sds_create_interface_type (
   Pcte_object_reference       sds,
   Pcte_name                   local_name,
   Pcte_types_names_in_sds     parents,
   Pcte_types_names_in_sds     new_operations,
   Pcte_type_name_in_sds       new_interface
);
```

(8) /* The effect of not providing the optional parameter *local_name* to the abstract operation */
/* is achieved by specifying **local_name** as NULL. */

/* G.3.2.6 SDS_CREATE_OBJECT_PARAMETER_TYPE */

(9)
```
Pcte_error_type Pcte_sds_create_object_parameter_type (
   Pcte_object_reference    sds,
   Pcte_name                local_name,
   Pcte_type_name           object_type,
   Pcte_type_name           new_parameter
);
```

(10) /* The effect of not providing the optional parameter *local_name* to the abstract operation */
/* is achieved by specifying **local_name** as NULL. */

/* G.3.2.7 SDS_CREATE_OPERATION_TYPE */

(11)
```
Pcte_error_type Pcte_sds_create_operation_type (
   Pcte_object_reference       sds,
   Pcte_name                   local_name,
   Pcte_types_names_in_sds     parameters,
   Pcte_type_name_in_sds       return_value,
   Pcte_type_name_in_sds       new_operation
);
```

(12) /* The effect of not providing the optional parameter *local_name* to the abstract operation */
/* is achieved by specifying **local_name** as NULL. */

/* G.3.2.8 SDS_IMPORT_INTERFACE_TYPE */

(13)
```
Pcte_error_type Pcte_sds_import_interface_type (
   Pcte_object_reference       to_sds,
   Pcte_object_reference       from_sds,
   Pcte_type_name_in_sds       type,
   Pcte_name                   local_name,
   Pcte_interface_scope        import_scope
);
```

(14) /* The effect of not providing the optional parameter *local_name* to the abstract operation */
/* is achieved by specifying **local_name** as NULL. */

```
/* G.3.2.9 SDS_IMPORT_OPERATION_TYPE */
```

(15)
```
Pcte_error_type Pcte_sds_import_operation_type (
    Pcte_object_reference        to_sds,
    Pcte_object_reference        from_sds,
    Pcte_type_name_in_sds        type,
    Pcte_name                    local_name
);
```

(16) /* The effect of not providing the optional parameter *local_name* to the abstract operation */
/* is achieved by specifying **local_name** as NULL. */

```
/* G.3.2.10 SDS_UNAPPLY_INTERFACE_TYPE */
```

(17)
```
Pcte_error_type Pcte_sds_unapply_interface_type (
    Pcte_object_reference        sds,
    Pcte_type_name_in_sds        interface_type,
    Pcte_type_name_in_sds        type
);
```

```
/* G.3.2.11 SDS_UNAPPLY_OPERATION_TYPE */
```

(18)
```
Pcte_error_type Pcte_sds_unapply_operation_type (
    Pcte_object_reference        sds,
    Pcte_type_name_in_sds        operation_type,
    Pcte_type_name_in_sds        type
);
```

(19)
```
#endif
```

Index of abstract operations

Index of C subprograms

Index of C datatypes

ECMA

114 Rue du Rhône
CH-1204 Geneva
Switzerland