INTERNATIONAL STANDARD

ISO 10303-24

First edition
2001-12-15

Industrial automation systems and integration — Product data representation and exchange —

Part 24: Implementation methods: C language binding of standard data access interface

Systèmes d'automatisation industrielle et intégration — Représentation et échange de données de produits —

Partie 24: Méthode de mise en application: Liant de langage C à l'interface d'accès aux données normalisées

Reference number
ISO 10303-24:2001(E)

© ISO 2001

Contents

page

1 Scope . . . 1

2 Normative references . . . 1

3 Terms, definitions, and abbreviations . . . 2
- 3.1 Terms defined in ISO 10303-1 . . . 2
- 3.2 Terms defined in ISO 10303-11 . . . 2
- 3.3 Terms defined in ISO 10303-22 . . . 3
- 3.4 Other definitions . . . 3
- 3.5 Abbreviations . . . 4

4 Overview of the C language late binding of SDAI . . . 5
- 4.1 Language bindings . . . 5
- 4.2 Conformance . . . 5
- 4.3 Use of late binding functions . . . 5
  - 4.3.1 Invalid parameter values . . . 5
  - 4.3.2 Error handling . . . 5
  - 4.3.3 Memory management . . . 5
  - 4.3.4 The SDAI header file . . . 6
  - 4.3.5 Macros . . . 6
- 4.4 Naming and typographical conventions . . . 6

5 Constants and data type definitions . . . 7
- 5.1 Standard error codes . . . 7
- 5.2 EXPRESS constants . . . 7
- 5.3 EXPRESS data types . . . 7
  - 5.3.1 Bit data type . . . 7
  - 5.3.2 EXPRESS simple data types . . . 8
  - 5.3.3 Enumeration data type . . . 9
  - 5.3.4 Select data type . . . 9
  - 5.3.5 Entity data type . . . 10
  - 5.3.6 Aggregate data types . . . 10
- 5.4 SDAI data types . . . 10
  - 5.4.1 SDAI primitive data types . . . 10
  - 5.4.2 SDAI entity data types . . . 11
  - 5.4.3 Iterator data type . . . 13
  - 5.4.4 Non-persistent list data type . . . 13
  - 5.4.5 Query source data type . . . 13
  - 5.4.6 SDAI access type data type . . . 13
- 5.5 C late binding-specific data types . . . 13
  - 5.5.1 Attribute data block data type . . . 14
  - 5.5.2 Aggregate index data type . . . 14
  - 5.5.3 Error code data type . . . 14
  - 5.5.4 Error handler data type . . . 14
  - 5.5.5 Transaction commit mode data type . . . 15
  - 5.5.6 NULL identifier data type . . . 15

6 C late binding functions of the SDAI operations . . . 15
6.1 Environment operations . . . 16
6.1.1 Open session . . . 16
6.1.2 C late binding specific arithmetic operations . . . 16
6.1.3 C late binding specific error handling operations . . . 17
6.1.4 C late binding specific instance operations . . . 18
6.2 Session operations . . . 19
6.2.1 Record event . . . 19
6.2.2 Set event recording . . . 19
6.2.3 Close session . . . 20
6.2.4 Open repository . . . 20
6.2.5 Start transaction read-write or read-only access . . . 21
6.2.6 Break transaction . . . 22
6.2.7 End transaction access . . . 22
6.2.8 Create non-persistent list . . . 23
6.2.9 Delete non-persistent list . . . 23
6.2.10 SDAI query . . . 24
6.2.11 C late binding specific recording operations . . . 25
6.2.12 C late binding specific attribute data block operations . . . 26
6.3 Repository operations . . . 29
6.3.1 Create SDAI-model . . . 29
6.3.2 Create schema instance . . . 30
6.3.3 Close repository . . . 31
6.4 Schema instance operations . . . 31
6.4.1 Delete schema instance . . . 31
6.4.2 Rename schema instance . . . 32
6.4.3 Add SDAI-model . . . 33
6.4.4 Remove SDAI-model . . . 34
6.4.5 Validate global rule . . . 34
6.4.6 Validate uniqueness rule . . . 35
6.4.7 Validate instance reference domain . . . 36
6.4.8 Validate schema instance . . . 37
6.4.9 Is validation current . . . 38
6.4.10 Schema instance operations for convenience . . . 39
6.5 SDAI-model operations . . . 40
6.5.1 Delete SDAI-model . . . 40
6.5.2 Rename SDAI-model . . . 41
6.5.3 Start SDAI-model access . . . 42
6.5.4 Promote SDAI-model to read-write access . . . 42
6.5.5 End SDAI-model access . . . 43
6.5.6 Get entity definition . . . 44
6.5.7 Create entity instance . . . 44
6.5.8 Undo changes . . . 45
6.5.9 Save changes . . . 46
6.5.10 SDAI-model operations for convenience . . . 46
6.6 Scope operations . . . 48
6.6.1 Add to scope . . . 48
6.6.2 Is scope owner . . . 49
6.6.3 Get scope . . . 49

6.6.4 Remove from scope . . . 50
6.6.5 Add to export list . . . 51
6.6.6 Remove from export list . . . 51
6.6.7 Scoped delete . . . 52
6.6.8 Scoped copy in same SDAI-model . . . 53
6.6.9 Scoped copy to other SDAI-model . . . 53
6.6.10 Validate scope reference restrictions . . . 54
6.6.11 Scope operations for convenience . . . 55
6.7 Type operations . . . 57
6.7.1 Get complex entity definition . . . 57
6.7.2 Is subtype of . . . 58
6.7.3 Is SDAI subtype of . . . 59
6.7.4 Is domain equivalent with . . . 59
6.7.5 Type operations for convenience . . . 60
6.8 Entity instance operations . . . 61
6.8.1 Get attribute . . . 61
6.8.2 Test attribute . . . 62
6.8.3 Find entity instance SDAI-model . . . 63
6.8.4 Get instance type . . . 64
6.8.5 Is instance of . . . 64
6.8.6 Is kind of . . . 65
6.8.7 Is SDAI kind of . . . 66
6.8.8 Find entity instance users . . . 67
6.8.9 Find entity instance usedin . . . 68
6.8.10 Get attribute value bound . . . 69
6.8.11 Find instance roles . . . 70
6.8.12 Find instance data types . . . 70
6.8.13 Entity instance operations for convenience . . . 71
6.9 Application instance operations . . . 73
6.9.1 Copy application instance in same SDAI-model . . . 73
6.9.2 Copy application instance to other SDAI-model . . . 74
6.9.3 Delete application instance . . . 75
6.9.4 Put attribute . . . 75
6.9.5 Unset attribute value . . . 76
6.9.6 Create aggregate instance . . . 77
6.9.7 Create aggregate instance ADB . . . 78
6.9.8 Get persistent label . . . 79
6.9.9 Get session identifier . . . 80
6.9.10 Get description . . . 80
6.9.11 Validate where rule . . . 81
6.9.12 Validate required explicit attributes assigned . . . 82
6.9.13 Validate inverse attributes . . . 83
6.9.14 Validate explicit attributes references . . . 84
6.9.15 Validate aggregates size . . . 85
6.9.16 Validate aggregates uniqueness . . . 85
6.9.17 Validate array not optional . . . 86
6.9.18 Validate string width . . . 87
6.9.19 Validate binary width . . . 88
6.9.20 Validate real precision . . . 89

6.9.21 Application instance operations for convenience 90
6.10 Entity instance aggregate operations 91
6.10.1 Get member count 91
6.10.2 Is member 92
6.10.3 Create iterator 93
6.10.4 Delete iterator 94
6.10.5 Beginning 94
6.10.6 Next 95
6.10.7 Get current member 95
6.10.8 Get value bound by iterator 96
6.10.9 Get lower bound 97
6.10.10 Get upper bound 98
6.11 Application instance aggregate operations 99
6.11.1 Create aggregate instance as current member 99
6.11.2 Put current member 100
6.11.3 Remove current member 100
6.12 Application instance unordered collection operations 101
6.12.1 Add unordered 101
6.12.2 Create aggregate instance unordered 102
6.12.3 Remove unordered 103
6.13 Entity instance ordered collection operations 104
6.13.1 Get by index 104
6.13.2 End 105
6.13.3 Previous 105
6.13.4 Get value bound by index 106
6.14 Application instance ordered collection operations 107
6.14.1 Put by index 107
6.14.2 Create aggregate instance by index 108
6.15 Entity instance array operations 109
6.15.1 Test by index 109
6.15.2 Test current member 109
6.15.3 Get lower index 110
6.15.4 Get upper index 111
6.16 Application instance array operations 112
6.16.1 Unset value by index 112
6.16.2 Unset value current member 112
6.16.3 Reindex array 113
6.16.4 Reset array index 114
6.17 Application instance list operations 114
6.17.1 Add before current member 114
6.17.2 Add after current member 115
6.17.3 Add by index 116
6.17.4 Create aggregate instance before current member 117
6.17.5 Create aggregate instance after current member 118
6.17.6 Add aggregate instance by index 119
6.17.7 Remove by index 120
6.18 C late binding specific SELECT TYPE operations 120
6.18.1 Put ADB type path 121
6.18.2 Get ADB type path 121

6.18.3 Validate type path 122

Annex A (normative) Information object registration 124

Annex B (informative) The C late binding header include file <sdai.h> 125

Index 135

Tables page

Table 1 - SDAI C late binding error indicators 7
Table 2 - EXPRESS built-in constants 8
Table 3 - SDAI primitive data types mapped to C late binding 11
Table 4 - SDAI entity data types mapped to C late binding 12

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO 10303 may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

International Standard ISO 10303-24 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

This International Standard is organized as a series of parts, each published separately. The structure of this International Standard is described in ISO 10303-1.

Each part of this International Standard is a member of one of the following series: description methods, implementation methods, conformance testing methodology and framework, integrated generic resources, integrated application resources, application protocols, abstract test suites, application interpreted constructs, and application modules. This part is a member of the implementation methods series.

A complete list of parts of ISO 10303 is available from the Internet:

    <http://www.nist.gov/sc4/editing/step/titles/>

Annex A forms a normative part of this part of ISO 10303. Annex B is for information only.

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation of product information and for the exchange of product data. The objective is to provide a neutral mechanism capable of describing products throughout their life cycle. This mechanism is suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases, and as a basis for archiving.

This part of ISO 10303 specifies a C programming language late binding of capability specified in ISO 10303-22, the standard data access interface (SDAI). The SDAI defines a data access interface to data defined using ISO 10303-11 (EXPRESS). The SDAI specifies operations that give the application programmer the capability to manipulate data through an interface based upon its description in the defining schema or schemas. This part of ISO 10303 specifies manifestation of that interface in the C programming language that is independent of the EXPRESS data definitions being manipulated. The standardization of a data access interface along with data definitions facilitates integration of different software components from different vendors.

The document is structured corresponding to ISO 10303-22. The major subdivisions in this part of ISO 10303 are:

— Clause 4 is an overview of the C language late binding to the SDAI. It specifies the requirements common to all C language late binding functions.

— Clause 5 specifies the C language late bindings to the EXPRESS and binding specific constants and data types.

— Clause 6 specifies the C language late binding functions to the SDAI operations to handle the programming environment.

— The specification of the C language late binding functions for the SDAI operations follows the categories defined in ISO 10303-22 clause 10.

Computer application systems are implemented using computing languages. Since there are many computing languages, many SDAI language bindings are possible. Additional SDAI language bindings are specified as other parts of ISO 10303 within the implementation method series.

Implementations of this part of ISO 10303 are not required to support the complete set of capabilities specified in ISO 10303-22. Specific sets of capability are grouped into implementation classes. The implementation classes against which conformance may be claimed are defined in ISO 10303-22 clause 13.

Industrial automation systems and integration — Product data representation and exchange — Part 24: Implementation methods: C language binding of standard data access interface

1 Scope

This part of ISO 10303 specifies a C programming language late binding of the capability specified in ISO 10303-22 - Standard data access interface (SDAI). This binding is a late binding and as such, none of the constants, data types, and functions depend on the application schema being accessed.

The following are within the scope of this part of ISO 10303:

— access to and manipulation of data types and entities which are specified in ISO 10303-22;

— convenience functions suitable to this language binding;

— late binding requirements specified in ISO 10303-22.

The following are outside the scope of this part of ISO 10303:

— memory arrangement of data structures used by implementations of this part of ISO 10303;

— early binding requirements as specified in ISO 10303-22;

— all items listed as out of scope in ISO 10303-22.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO 10303. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO 10303 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 9899:1999, *Programming languages — C*

ISO/IEC 8824-1:1998, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation*

ISO 10303-1:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles*

ISO 10303-11:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual*

ISO 10303-21:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure*

ISO 10303-22:1998, *Industrial automation systems and integration - Product data representation and exchange - Part 22: Implementation methods: Standard data access interface*

3 Terms, definitions, and abbreviations

3.1 Terms defined in ISO 10303-1

For the purposes of this part of ISO 10303, the following terms defined in ISO 10303-1 apply.

— application;

— application protocol;

— conformance testing;

— data;

— implementation method;

— information;

— model.

3.2 Terms defined in ISO 10303-11

For the purposes of this part of ISO 10303, the following terms defined in ISO 10303-11 apply.

— complex entity data type;

— data type;

— entity;

— entity data type;

— entity instance;

— instance.

3.3 Terms defined in ISO 10303-22

For the purposes of this part of ISO 10303, the following terms defined in ISO 10303-22 apply.

— application schema;

— constraint;

— identifier;

— iterator;

— implementation class;

— repository;

— schema instance;

— SDAI language binding;

— SDAI-model;

— session;

— validation.

3.4 Other definitions

For the purposes of this part of ISO 10303, the following definitions apply:

3.4.1
attribute data block
a C structure containing both a value and the data type of the value that is accessed through a handle.

3.4.2
function
a C language late binding specific interpretation of an SDAI operation, a combination of several SDAI operations or an operation unique to this binding.

3.4.3
function prototype
the definition of a C programming language function in an include file.

3.4.4
handle C type
a function parameter that is a C language pointer type containing the address of a datum or a structured data.

3.5 Abbreviations

For the purposes this part of ISO 10303, the following abbreviations apply:

aggr	Aggregate
app	Application
attr	Attribute
ADB	Attribute Data Block
BN	By name
Deq	Domain equivalent
Enum	Enumeration
Id	Identifier
Itr	Iterator
NPL	Non-persistent List
Rep	Repository
RO	Read only
RW	Read write
SDAI	Standard Data Access Interface
Trx	Transaction
Uni	Uniqueness

4 Overview of the C language late binding of SDAI

4.1 Language bindings

ISO 10303-22 specifies operations independently of any programming language. Language bindings of the operations are developed for programming languages to define the capability required of conforming implementations. Two types of language bindings are identified: late bindings and early bindings. The concept of language bindings is defined in ISO 10303-22 clause 4. This part of ISO 10303 specifies a C language late binding of the SDAI operations.

This part of ISO 10303 supports all of the functionality defined in ISO 10303-22. There is not a one-to-one correspondence between the operations described in ISO 10303-22 and the functions defined in this part of ISO 10303. This part of ISO 10303 extends the functionality defined in ISO 10303-22 to provide more efficient or convenient operations.

4.2 Conformance

An implementation of this part of ISO 10303 shall conform to an implementation class as specified in ISO 10303-22 clause 13. The implementation shall support all C language binding functions whose original specification in ISO 10303-22 contains an operation required by the implementation level and shall support all convenience functions defined in this part of ISO 10303.

4.3 Use of late binding functions

4.3.1 Invalid parameter values

If a parameter to a C late binding function has an invalid value (such as a value outside the domain of the function, a pointer outside the address space of the program, or a NULL pointer), the behaviour of the function is not specified in this part of ISO 10303.

4.3.2 Error handling

In the event an error is detected during the execution of a function, the values of the input parameters, the state of the implementation, and the application data managed by the implementation shall be unchanged, except in the event of a fatal underlying system error where the outcome is dependent on system and implementation design. Whether the output parameters are affected in the event of an error is left to the implementation.

4.3.3 Memory management

When applicable, functions are strongly typed to return either a designated value or an instance identifier. The function parameters do not include output parameters except where required to return untyped data in application managed storage. The output parameters are typed `void*` to accept arbitrarily typed pointers to output buffers for attribute and aggregate member values or ADBs passed from the implementation to the application program.

The SDAI implementation shall be responsible for allocating and deallocating memory for attribute values of the EXPRESS data types BINARY, ENUMERATION, and STRING, and the application program is responsible for the contents of that memory. The SDAI implementation shall allocate memory in a fashion such that it is large enough for any character or binary string to be read. The value of that memory shall be unchanged until the next time a character or binary string is read, or until the end of the session.

If the SDAI implementation supports the computation of EXPRESS DERIVE attributes, the implentation shall allocate and deallocate memory for the computed value. The value of that memory shall be unchanged until the next time a DERIVE attribute is read, or until the end of the session.

If the SDAI implementation supports the computation of EXPRESS INVERSE attributes, the implentation shall allocate and deallocate memory for the NPL containing the result. The value of the NPL shall be unchanged until the next time an INVERSE attribute is read, or until the end of the session.

4.3.4 The SDAI header file

An implementation shall provide a C late binding program header file, named `sdai.h`, for inclusion into the application program by the C preprocessor directive `#include <sdai.h>`. This header file shall contain all the declarations of types, constants, and functions defined in this part of ISO 10303. An example header file is provided in annex B.

4.3.5 Macros

Any SDAI operation may be implemented as a macro defined in the `sdai.h` header file. Any invocation of an operation that is implemented as a macro shall expand resulting in each parameter being evaluated exactly once, fully protected by parentheses where required.

4.4 Naming and typographical conventions

In this part of ISO 10303, C language function, type, and constant names are typeset in a monospace font to distinguish them from ordinary text.

The prefix `sdai` is used for all C programming language function, type, and constant names. Case is used to delimit separate words or parts of identifiers and to differentiate among function, type, and constant names as follows:

— function names are prefixed by a lowercase `sdai` and each word in the function name starts with an uppercase letter;

EXAMPLE 1 `sdaiOpenSession` for the SDAI Open Session operation.

— type names are prefixed by `Sdai` starting with an uppercase letter s and each word in the type name starts with an uppercase letter;

EXAMPLE 2 `SdaiNamedType` for EXPRESS named types.

— constant names are prefixed by a lowercase `sdai` and the constant name is all uppercase letters.

EXAMPLE 3 `sdaiRO` for read-only.

5 Constants and data type definitions

5.1 Standard error codes

ISO 10303-22 clause 11 defines a set of SDAI error indicators. These error indicators with the addition of the C language binding specific error indicators defined in Table 1 define the set of error indicators mapped into C programming language constants to which all implementations shall support access via the Error Query function (see 6.1.3.1). The constant name is the name of the SDAI error indicator prefixed by `sdai`.

EXAMPLE `sdaiMO_NEXS` for SDAI-model does not exist.

The error code constants shall be included in the `sdai.h` header file. The value and type of the error code constants are not defined in ISO 10303-22 nor in this part of ISO 10303. The error code value and error base specified need only be supported as attributes of the error event defined in ISO 10303-22: 7.4.7.

Table 1 - SDAI C late binding error indicators

Error indicator	Description	Error code	Error base
`NO_ERR`	No error	Not applicable	
`AB_NEXS`	Attribute data block does not exist	2400	

5.2 EXPRESS constants

Table 2 specifies the EXPRESS built-in constants represented by C language constants or macros. These built in constants shall be included in the `sdai.h` header file.

5.3 EXPRESS data types

5.3.1 Bit data type

The C late binding specific data type `SdaiBit` is used to build `SdaiBinary` values. The C late binding data type `SdaiBinary` represents the EXPRESS BINARY data type. `SdaiBit` has two distinct values `sdaiBIT0` and `sdaiBIT1`. The `SdaiBit` type shall be defined as the C late binding data type:

```
typedef unsigned char SdaiBit;
```

Table 2 - EXPRESS built-in constants

EXPRESS constant	C late binding name	C data type	Value
CONST_E	`sdaiE`	`double`	specified in ISO 10303-11 with the precision not specified in this part of ISO 10303
PI	`sdaiPI`	`double`	specified in ISO 10303-11 with the precision not specified in this part of ISO 10303
FALSE	`sdaiFALSE`	`int`	0
TRUE	`sdaiTRUE`	`int`	1
UNKNOWN	`sdaiUNKNOWN`	`int`	2
BINARY bit 0	`sdaiBIT0`	`SdaiBit`	not specified in this part of ISO 10303
BINARY bit 1	`sdaiBIT1`	`SdaiBit`	not specified in this part of ISO 10303

5.3.2 EXPRESS simple data types

5.3.2.1 EXPRESS INTEGER data type

The EXPRESS INTEGER data type shall be represented by the C late binding data type:

```
/* C late binding simple data types: */
typedef long      SdaiInteger;
```

5.3.2.2 EXPRESS REAL data type

The EXPRESS REAL data type shall be represented by the C late binding data type:

```
typedef double    SdaiReal;
```

5.3.2.3 EXPRESS NUMBER data type

The EXPRESS NUMBER data type shall be represented by the C late binding data type:

```
typedef SdaiReal  SdaiNumber;
```

5.3.2.4 EXPRESS BOOLEAN data type

The EXPRESS BOOLEAN data type shall be represented by the C late binding data type:

```
typedef int       SdaiBoolean;
```

`SdaiBoolean` shall be compatible with `sdaiFALSE` and `sdaiTRUE` of type `SdaiLogical`.

5.3.2.5 EXPRESS LOGICAL data type

The EXPRESS LOGICAL data type shall be represented by the C late binding data type:

```
typedef int       SdaiLogical;
```

`SdaiLogical` shall be compatible with `sdaiFALSE` and `sdaiTRUE` and the boolean operations of the C programming language. Comparison operations (=,>,<) are supported by a late binding function (see 6.1.2.1).

5.3.2.6 EXPRESS STRING data type

The EXPRESS STRING data type shall be represented by the C late binding data type:

```
typedef char     *SdaiString;
```

5.3.2.7 EXPRESS BINARY data type

The EXPRESS BINARY data type shall be represented by the C late binding data type:

```
typedef SdaiBit  *SdaiBinary;
```

Indexing using `[]` and automatic conversion between `SdaiBinary` and `SdaiInteger` shall be supported.

5.3.3 Enumeration data type

The EXPRESS ENUMERATION data type shall be represented by the C late binding data type `SdaiEnum`. This data type shall be compatible with `char*` and implementations shall support passing enumeration values as lower case C string literals:

```
/* enumeration data type: */
typedef char     *SdaiEnum;
```

5.3.4 Select data type

The value of an attribute or aggregate member that has as its type domain an EXPRESS SELECT data type may be required to be represented as an SDAI **select_value** as specified in ISO 10303-22: 9.4.8. The SDAI **select_value** is represented by the C late binding data type `SdaiADB`, an attribute data block. When the type of the value is not ambiguous, it need not require representation as a **select_value** and may be represented by the appropriate C type.

EXAMPLE If the value is an entity instance identifier then it is not necessary to represent it as a **select_value**.

5.3.5 Entity data type

The value of an attribute or aggregate element that is an instance of an EXPRESS ENTITY type is represented by an implementation specific handle named `SdaiId`. The handle serves as the identifier of the instance. Identifiers are not persistent. Identifiers shall be unique globally over all types of instances and unchanging within an SDAI session for any particular instance.

```
/* entity instance identifier type: */
typedef SdaiId    SdaiInstance;
```

Explicit referencing using `SdaiInstance` need not be supported by an implementation.

5.3.6 Aggregate data types

A generalization of the EXPRESS aggregate data types is represented by data type **aggregate_instance** and its subtypes defined in ISO 10303-22 clause 9. The C late binding representation of any aggregate instance is defined by the instance identifier type `SdaiInstance` and is named `SdaiAggr`. Aggregate instance identifiers shall be unique during an SDAI session.

```
/* aggregate data types: */
typedef SdaiInstance         SdaiAggr;
```

The C late binding aggregate data types form a hierarchy as follows:

```
typedef SdaiAggr             SdaiOrderedAggr;
typedef SdaiAggr             SdaiUnorderedAggr;
typedef SdaiOrderedAggr      SdaiArray;
typedef SdaiOrderedAggr      SdaiList;
typedef SdaiUnorderedAggr    SdaiSet;
typedef SdaiUnorderedAggr    SdaiBag;
```

5.4 SDAI data types

ISO 10303-22 specifies many data types in the schemas it defines. These data types are used as parameters to the SDAI operations. The mapping of these data types into the C language are specified in this subclause.

5.4.1 SDAI primitive data types

The SDAI primitive data types of values given for or expected from a particular attribute or aggregate element are specified in the SDAI parameter data schema (see ISO 10303-22 clause 9). The SDAI primitive types are represented in the C language late binding as described in Table 3.

The SDAI primitive type (see ISO 10303-22: 9.3.1) is used to specify an attribute or aggregate element value. It is represented in the C binding by the type `SdaiPrimitiveType`. It is an enumeration defined as:

Table 3 - SDAI primitive data types mapped to C late binding

SDAI primitive data type	C late binding type
binary_value	`SdaiBinary`
boolean_value	`SdaiBoolean`
enumeration_value	`SdaiEnum`
integer_value	`SdaiInteger`
logical_value	`SdaiLogical`
number_value	`SdaiNumber`
real_value	`SdaiReal`
select_value	`SdaiADB`
string_value	`SdaiString`
entity_instance	`SdaiInstance`
aggregate_instance	`SdaiAggr`

```
/* attribute type data type: */
typedef enum {
      sdaiADB,      sdaiAGGR,     sdaiBINARY,   sdaiBOOLEAN,
      sdaiENUM,     sdaiINSTANCE, sdaiINTEGER,  sdaiLOGICAL,
      sdaiNOTYPE,   sdaiNUMBER,   sdaiREAL,     sdaiSTRING
} SdaiPrimitiveType;
```

5.4.2 SDAI entity data types

References to instances of EXPRESS entity data types managed by the implementation are made in the C late binding by instance identifiers. ISO 10303-22 defines entity types in the SDAI dictionary schema, the SDAI session schema, the SDAI population schema, and the SDAI parameter data schema (see ISO 10303-22 clauses 6 through 9). For those entity types that are strongly typed, Table 4 specifies their representation in the C language late binding. ISO 10303-22 entity types from the SDAI dictionary schema, SDAI session schema and SDAI population schema for which a representation is not specified explicitly in this part of ISO 10303 are represented by the C binding type `SdaiInstance`.

Instances of entity data types managed by the implementation are referenced in the C late binding by instance identifiers of the type `SdaiInstance`:

```
/* SDAI instance identifier types: */
typedef SdaiInstance         SdaiAppInstance;
typedef SdaiInstance         SdaiModel;
typedef SdaiInstance         SdaiRep;
typedef SdaiInstance         SdaiSession;
typedef SdaiInstance         SdaiAttr;
typedef SdaiAttr             SdaiExplicitAttr;
```

Table 4 - SDAI entity data types mapped to C late binding

SDAI Schema	SDAI entity data type	C late binding type
Dictionary	attribute	`SdaiAttr`
Dictionary	defined_type	`SdaiDefinedType`
Dictionary	entity_definition	`SdaiEntity`
Dictionary	explicit_attribute	`SdaiExplicitAttr`
Dictionary	global_rule	`SdaiGlobalRule`
Dictionary	named_type	`SdaiNamedType`
Dictionary	schema_definition	`SdaiSchema`
Dictionary	uniqueness_rule	`SdaiUniRule`
Dictionary	where_rule	`SdaiWhereRule`
Dictionary	global_rule	`SdaiGlobalRule`
Parameter data	aggregate_instance	`SdaiAggr`
Parameter data	application_instance	`SdaiAppInstance`
Parameter data	array_instance	`SdaiArray`
Parameter data	bag_instance	`SdaiBag`
Parameter data	entity_instance	`SdaiInstance`
Parameter data	list_instance	`SdaiList`
Parameter data	non_persistent_list_instance	`SdaiNPL`
Parameter data	ordered_collection	`SdaiOrderedAggr`
Parameter data	set_instance	`SdaiSet`
Parameter data	unordered_collection	`SdaiUnorderedAggr`
Population	schema_instance	`SdaiSchemaInstance`
Population	scope	`SdaiScope`
Population	sdai_model	`SdaiModel`
Session	sdai_repository	`SdaiRep`
Session	sdai_session	`SdaiSession`
Session	sdai_transaction	`SdaiTrx`

```
typedef SdaiInstance         SdaiNamedType;
typedef SdaiNamedType        SdaiEntity;
typedef SdaiNamedType        SdaiDefinedType;
typedef SdaiInstance         SdaiWhereRule;
typedef SdaiInstance         SdaiUniRule;
typedef SdaiInstance         SdaiGlobalRule;
typedef SdaiInstance         SdaiSchema;
typedef SdaiInstance         SdaiScope;
typedef SdaiInstance         SdaiSchemaInstance;
typedef SdaiInstance         SdaiTrx;
```

5.4.3 Iterator data type

The SDAI iterator data type (see ISO 10303-22: 9.4.1) providing access to aggregate members is represented by an implementation specific handle named `SdaiItrId`:

```
/* SDAI iterator identifier type: */
typedef SdaiItrId            SdaiIterator;
```

5.4.4 Non-persistent list data type

The SDAI non persistent list instance data type (see ISO 10303-22: 9.4.18) is represented in the C language binding as `SdaiNPL`. NPLs may be accessed by any C late binding function that has a parameter of the type `SdaiList`.

```
/* Non-persistent list data type: */
typedef SdaiList             SdaiNPL;
```

5.4.5 Query source data type

The SDAI query source data type (see ISO 10303-22: 9.3.12) is represented in the C language binding as `SdaiQuerySourceType`. It is used to specify the domain over which an SDAI query operation is executed.

```
/* Query source data type: */
typedef enum {
      sdaiAGGR, sdaiMODEL, sdaiREP, sdaiSCHEMAINSTANCE
} SdaiQuerySourceType;
```

5.4.6 SDAI access type data type

The SDAI access type data type (see ISO 10303-22: 7.3.1) is represented in the C language binding as `SdaiAccessMode`. It is used to specify the mode when starting access to an SDAI-model or starting a transaction. It is an enumeration type consisting of two values, representing read-only and read-write access, and is defined as:

```
/* access mode data type: */
typedef enum {
      sdaiRO, sdaiRW
} SdaiAccessMode;
```

5.5 C late binding-specific data types

This part of ISO 10303 requires several C data types in addition to those required by ISO 10303-22. This subclause defines those additional types.

5.5.1 Attribute data block data type

An attribute data block represents a value together with its data type. The ADB data type `SdaiADB` is used in functions when the type of a value is decided only at the instance level, for example, when reading the attribute value of an attribute that may be one of several potential data types. The ADBs are used for get and put functions of attribute and aggregate leaf element values. The ADBs are also used to set and read the type path information necessary as the result of some EXPRESS SELECT TYPEs. As ADB identifiers need not be persistent between SDAI sessions, functions comparing attribute values or aggregate members represented by ADBs shall only consider the data value and data type in the comparison.

The C late binding type of the type `SdaiADB` is represented by an implementation specific handle named `SdaiADBId`:

```
/* C late binding ADB identifier type: */
typedef SdaiADBId            SdaiADB;
```

Access to the contents of an ADB is made using the C language late binding specific ADB and SELECT TYPE functions.

5.5.2 Aggregate index data type

The data type `SdaiAggrIndex` is used to represent aggregation indices. The C late binding type of `SdaiAggrIndex` is represented by an implementation specific handle named `SdaiIndexId`:

```
/* aggregate index data type: */
typedef SdaiIndexId          SdaiAggrIndex;
```

5.5.3 Error code data type

The data type `SdaiErrorCode` is used to represent the C late binding error constants (see 5.1). The C late binding type of `SdaiErrorCode` is represented by a handle named `SdaiErrorId`:

```
/* error code data type: */
typedef SdaiErrorId          SdaiErrorCode;
```

5.5.4 Error handler data type

`SdaiErrorHandler` is the type of all error handling functions specified by the application to be used by the implementation upon error detection. These functions shall accept a single `SdaiErrorCode` parameter. The prototype is defined as:

```
/* error handler data type: */
typedef void (*SdaiErrorHandler)(SdaiErrorCode);
```

5.5.5 Transaction commit mode data type

SdaiCommitMode is used to specify whether the most recent effect of changes made during a particular transaction is to be committed or not. The transaction commit mode data type shall be mapped as an enumeration type defined as:

```
/* transaction commit mode data type: */
typedef enum {
      sdaiABORT, sdaiCOMMIT
} SdaiCommitMode;
```

5.5.6 NULL identifier data type

The sdaiNullId instance signifies the NULL identifier that may be compared with other instance identifiers to determine whether they have a valid value or not.

```
/* NULL identifier data type: */
typedef SdaiId               SdaiNullId;
```

6 C late binding functions of the SDAI operations

This clause defines the C late binding functions for the operations defined in ISO 10303-22. For operations specified in ISO 10303-22, the clause headings are the operation name as specified in ISO 10303-22. For operations defined in this part of ISO 10303, the clause headings are based upon the C function names but avoid abbreviations and specify a more complete name for the function. Each function is defined by the following as required:

— a description of the task performed by the function. For functions based on operations defined in ISO 10303-22, this description is incomplete in that the complete text describing the operation in ISO 10303-22 is not duplicated in this part of ISO 10303. Within this description, the operation name is based upon the C function name rather than the SDAI operation name;

— Prototype: /* the ANSI-C style function prototypes of the SDAI operations */;

— Input: the parameters required to be specified prior to the execution of the function;

— Output: the parameters made available to the application after the successful execution of the function;

— Return: the function value made available to the application after the successful execution of the function;

— Possible error indicators: the error code constant values that may be the SdaiErrorCode return of the Error Query function after the unsuccessful execution of a C binding function;

— Original specification in ISO 10303-22: the clause containing the specification of the operation(s) in ISO 10303-22 upon which the function is based. If not specified, the function is a convenience function defined in this part of ISO 10303.

Certain functions need as an input some object that defines part of the operational environment, such as a repository. The object, as all instances, is identified by its instance identifier; however, an alternative function is provided that uses the name of the object as the input parameter. The name of such a function is to append `BN` to the name of the other function that performs the same operation using instance identifier as an input. For input or output parameters to a BN function that specify an EXPRESS identifier, the implementation need only support lower case letters (see ISO 10303-22: 6.3.6).

6.1 Environment operations

6.1.1 Open session

The Open Session function shall initiate the SDAI implementation and start a new SDAI session.

Prototype:

```
SdaiSession sdaiOpenSession (void);
```

Return:

In normal condition:	Session instance identifier.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_OPN`	Session open.
`sdaiSS_NAVL`	SDAI not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.3.1

6.1.2 C late binding specific arithmetic operations

6.1.2.1 Logical compare

The Logical Compare function shall test for the ordering of two values according to the ordering of the values of the EXPRESS LOGICAL data type.

Prototype:

```
int sdaiLogicalCompare (SdaiLogical value1, SdaiLogical value2);
```

Input:

value1:	`SdaiLogical`.
value2:	`SdaiLogical`.

Return:

In normal condition:	+1 if value1 is greater than value2; 0 if value1 is equal to value2; -1 if value1 is less than value2.

6.1.3 C late binding specific error handling operations

This subclause defines the functions for the purpose of handling the error resulting from the unsuccessful execution of a function. Error handling shall be managed via a default system error handling function, special error handling functions and the Error Query function. A last in, first out error handler stack shall be supported by implementations of this part of ISO 10303. Error handling functions are added to this stack by the Set Error Handler function and are removed from the stack by the Restore Error Handler function. If the error handler stack is empty, the default system error handling function shall be automatically invoked in the event an error occurs. If the error handler stack is not empty, the error handling function at the top of the stack shall be automatically invoked in the event an error occurs.

6.1.3.1 Error query

The Error Query function shall return the error code resulting from the C binding function that most recently executed unsuccessfully. After returning the error code, subsequent executions of the Error Query function shall return `sdaiNO_ERR` until another C language binding function executes unsuccessfully. Prior to executing the Open Session function, the Error Query function shall return `sdaiSS_NOPN`.

Prototype:

```
SdaiErrorCode sdaiErrorQuery (void);
```

Return:

Standard error code:	The error code of the most recent function that executed unsuccessfully.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.

6.1.3.2 Set error handler

The Set Error Handler function shall place the specified error handling function on the last in, first out error handler stack. This function shall accept the error code as the only parameter and may be executed before the Open Session function, during an SDAI session, and after the Close Session function. The system default error handler may be placed on the stack by specifying a NULL function. Multiple error handling functions may be placed on the stack. The most recently added error handling function is at the top of the stack and shall automatically be invoked when an error condition occurs.

Prototype:

```
void sdaiSetErrorHandler (SdaiErrorHandler function);
```

Possible error indicators:

`sdaiSY_ERR`	Underlying system error.

6.1.3.3 Restore error handler

The Restore Error Handler function shall remove the most recently added error handling function from the top of the last in, first out error handler stack. The next most recently added error handling function is left at the top of the error handler stack. This function has no effect if the error handler stack is empty.

Prototype:

```
SdaiErrorHandler sdaiRestoreErrorHandler (void);
```

Return:

If the stack was not empty:	a pointer to the top error handling function before removing it from the error handler stack.
If the stack was empty:	a NULL pointer.

Possible error indicators:

`sdaiSY_ERR`	Underlying system error.

6.1.4 C late binding specific instance operations

6.1.4.1 Is equal

The Is Equal function shall test whether the two specified SDAI identifiers are identical.

Prototype:

```
SdaiBoolean sdaiIsEqual (SdaiInstance instance1,
             SdaiInstance instance2);
```

Input:

instance1:	The first identifier in the comparison.
instance2:	The second identifier in the comparison.

Return:

In normal condition:	`sdaiTRUE` if instance1 and instance2 are equal; `sdaiFALSE` if instance1 and instance2 are not equal.

6.2 Session operations

6.2.1 Record event

The Record Event function shall append an event to the SDAI session events record.

Prototype:

```
void sdaiRecordEvent (SdaiSession session, SdaiString functionName,
             SdaiErrorCode error, SdaiString description);
```

Input:

session:	The identifier of the session in which the event takes place.
functionName:	An identifier for the function with which the event is associated.
error:	Error code for the error event.
description:	Description of the error event.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiER_NSET`	Event recording not set.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.1

6.2.2 Set event recording

The Set Event Recording function shall either activate event recording or inhibit event recording for the specified SDAI session.

Prototype:

```
SdaiBoolean sdaiSetEventRecording (SdaiSession session,
             SdaiBoolean setRec);
```

Input:

session:	The identifier of the session for which to activate or inhibit event recording.

setRec:	`sdaiTRUE` to activate event recording, `sdaiFALSE` to inhibit event recording.

Return:

In normal condition:	`sdaiTRUE` event recording is set as requested.
In error condition:	`sdaiFALSE` if event recording is not supported at all.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.2 and 10.4.3

6.2.3 Close session

The Close Session function shall terminate the specified SDAI session. Subsequent invocations of Error Query shall return `sdaiSS_NOPN` until the Open Session function executes successfully. After invoking this function, subsequent C late binding functions need no longer execute successfully.

Prototype:

```
void sdaiCloseSession (SdaiSession session);
```

Input:

session:	The identifier of the session to be closed.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.4

6.2.4 Open repository

The Open Repository function shall make the repository and its contents available to the session.

Prototype:

```
SdaiRep sdaiOpenRepository (SdaiSession session,
             SdaiRep repository);
SdaiRep sdaiOpenRepositoryBN (SdaiSession session,
             SdaiString repositoryName);
```

Input:

session:	Identifier of the session in which repository is to be opened.
repository:	Identifier of the repository.
repositoryName:	Name of the repository.

Return:

In normal condition:	Repository instance identifier.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiRP_NAVL`	Repository not available.
`sdaiRP_OPN`	Repository open.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.5

6.2.5 Start transaction read-write or read-only access

The Start Trx function shall a initiate a transaction with either a read-write or read-only access mode for subsequent functions.

Prototype:

```
SdaiTrx sdaiStartTrx (SdaiSession session, SdaiAccessMode mode);
```

Input:

session:	Identifier of the session whose transaction is to be started.
mode:	Access mode of the transaction to be started: `sdaiRW` for read-write access, `sdaiRO` for read-only access.

Return:

In normal condition:	Identifier of the started transaction.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.

`sdaiTR_EXS`	Transaction exists.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.6 and 10.4.7

6.2.6 Break transaction

The Break Trx function shall commit or abort changes made during a transaction. When the specified mode is `sdaiABORT`, this function shall abort changes made since the most recent Start Trx function where the Start Trx mode was specified as `sdaiRW` or since the most recent Break Trx where the Break Trx mode was specified as `sdaiCommit`. When the specified mode is `sdaiCOMMIT`, this function shall commit changes made since the most recent Start Trx function where the Start Trx mode was specified as `sdaiRW` or since the most recent Break Trx where the Break Trx mode was specified as `sdaiCommit`.

Prototype:

```
void sdaiBreakTrx (SdaiTrx transaction, SdaiCommitMode mode);
```

Input:

transaction:	Identifier of the active read-write access transaction.
mode:	Commit mode, can be: `sdaiABORT`, or `sdaiCOMMIT`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NEXS`	Transaction does not exist.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NAVL`	Transaction not available.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.8 and 10.4.9

6.2.7 End transaction access

The End Trx function shall terminate the specified transaction and either commit or abort all changes made since the most recent Start Transaction or Break Transaction function.

Prototype:

```
void sdaiEndTrx (SdaiTrx transaction, SdaiCommitMode mode);
```

Input:

transaction:	Identifier of the active read-write access transaction.
mode:	Commit mode: either `sdaiABORT`, or `sdaiCOMMIT`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NEXS`	Transaction does not exist.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NAVL`	Transaction not available.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.10 and 10.4.11

6.2.8 Create non-persistent list

The Create NPL function shall create a non-bounded, non-persistent list. NPLs shall be accessible by any C late binding function that has a parameter of the type `SdaiList`. An implementation of this part of ISO 10303 need not support the assignment of the identifier of a non-persistent list to an attribute of an entity instance or as an aggregate member.

Prototype:

```
SdaiNPL sdaiCreateNPL (void);
```

Return:

In normal condition:	Identifier of the newly created NPL.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.12

6.2.9 Delete non-persistent list

The Delete NPL function shall remove the specified non-persistent list from the SDAI session.

Prototype:

```
void sdaiDeleteNPL (SdaiNPL list);
```

Input:

list:	Identifier of a non-persistent list.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.13

6.2.10 SDAI query

The Query function shall determine those entity instances from the source domain that meet the specified criteria and append them to the specified pre-existing result NPL. The criteria valid for this function are limited to a subset of EXPRESS expressions.

Prototype:

```
SdaiInteger sdaiQuery (SdaiQuerySourceType sourceType,
             SdaiString criteria, SdaiInstance instance,
             SdaiNPL result, ...);
```

Input:

sourceType:	Type of the domain to be analyzed, with one of the following argument values: `sdaiAGGR`, `sdaiMODEL`, `sdaiREP`, or `sdaiSCHEMAINSTANCE`.
criteria:	The logical expression that defines the criteria to be evaluated.
instance:	The value for SELF in the case where the attribute being queried is a reference to an entity data type.
result:	Identifier of a pre-existing NPL to which the instance identifiers for those entity instances are appended meeting the specified criteria.
... :	Handle matching or convertible to the sourceType given as a function parameter with the specified late binding type.

Return:

In normal condition:	Number of entity instances meeting the specified criteria.
In error condition:	-1

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiEI_NVLD`	Entity instance invalid.
`sdaiVA_NVLD`	Value invalid.
`sdaiOP_NVLD`	Operator invalid.
`sdaiAT_NVLD`	Attribute invalid.
`sdaiVT_NVLD`	Value type invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.4.14

6.2.11 C late binding specific recording operations

6.2.11.1 Is recording on

The Is Recording On function shall indicate whether the session event recording is active or inhibited.

Prototype:

```
SdaiLogical sdaiIsRecordingOn (SdaiSession session);
```

Input:

session:	The identifier of the session test for event recording being active.

Return:

In normal condition:	`sdaiTRUE` if **session.recording_active** is TRUE; `sdaiFALSE` if **session. recording_active** is FALSE.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

6.2.12 C late binding specific attribute data block operations

6.2.12.1 Create ADB

The Create ADB function shall create an ADB and may set the type and value in the ADB. The Create Empty ADB function does not set the type and value in the newly created ADB. ADBs shall not contain other ADBs as values.

Prototype:

```
SdaiADB sdaiCreateADB (SdaiPrimitiveType valueType, ...);

SdaiADB sdaiCreateEmptyADB (void);
```

Input:

valueType:	One of the following values: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiINSTANCE`, or `sdaiAGGR`.
... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Return:

These functions shall return an ADB identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiVT_NVLD`	Value type invalid.
`sdaiSY_ERR`	Underlying system error.

6.2.12.2 Get ADB value

The Get ADB Value function shall get, and may convert, the value from the ADB. The type of the value to return shall be specified using the valueType parameter.

Prototype:

```
void *sdaiGetADBValue (SdaiADB block, SdaiPrimitiveType valueType,
             void *value);
```

Input:

block:	Attribute data block containing the value to be returned.

valueType:	One of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiNUMBER`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiINSTANCE`, or `sdaiAGGR`.
value:	Handle matching, or convertible to, the valueType.

Output:

value:	The handle filled with the primitive or identifier value returned from the ADB.

Return:

This function shall return the value argument filled with the primitive or identifier value returned from the ADB.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiAB_NEXS`	ADB does not exist.
`sdaiVT_NVLD`	Value type invalid.
`sdaiVA_NSET`	Value not set.
`sdaiSY_ERR`	Underlying system error.

6.2.12.3 Put ADB value

The Put ADB Value function shall set the type and value in the ADB. ADBs shall not contain other ADBs as values.

Prototype:

```
void sdaiPutADBValue (SdaiADB block,
             SdaiPrimitiveType valueType, ...);
```

Input:

block:	Attribute data block to be set.
valueType:	One of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiINSTANCE`, or `sdaiAGGR`.
... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the value type, and given as a function parameter with the specified C late binding type.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiAB_NEXS`	ADB does not exist.
`sdaiVT_NVLD`	Value type invalid.
`sdaiSY_ERR`	Underlying system error.

6.2.12.4 Get ADB type

The Get ADB Type function shall return the type of the value in the ADB.

Prototype:

```
SdaiPrimitiveType sdaiGetADBType (SdaiADB block);
```

Input:

block:	Attribute data block containing the type to be returned.

Return:

In normal condition:	The type of the primitive, or identifier value in the ADB block. The function shall return `sdaiNOTYPE` in case the ADB is empty. This function may return the following values if the ADB is not empty: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiINSTANCE`, or `sdaiAGGR`.
In error condition:	The value of `sdaiNOTYPE` is returned.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiAB_NEXS`	ADB does not exist.
`sdaiVA_NSET`	Optional value unset.
`sdaiSY_ERR`	Underlying system error.

6.2.12.5 Unset ADB

The Unset ADB function shall unset the type and value of the specified ADB. After invoking the Unset ADB function and before another type is set in the ADB, the Get ADB type function shall return `sdaiNOTYPE`. After invoking the Unset ADB function and before another value is set in the ADB, the Get ADB value function shall return the `sdaiVA_NSET` error.

Prototype:

```
void sdaiUnsetADB (SdaiADB block);
```

Input:

block:	Attribute data block whose value and type is to be unset.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiAB_NEXS`	ADB does not exist.
`sdaiSY_ERR`	Underlying system error.

6.2.12.6 Delete ADB

The Delete ADB function shall delete the specified ADB.

Prototype:

```
void sdaiDeleteADB (SdaiADB block);
```

Input:

block:	Attribute data block to be deleted.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiAB_NEXS`	ADB does not exist.
`sdaiSY_ERR`	Underlying system error.

6.3 Repository operations

6.3.1 Create SDAI-model

The Create Model function shall create a new SDAI-model based upon the specified schema in the specified repository.

Prototype:

```
SdaiModel sdaiCreateModel (SdaiRep repository, SdaiString modelName,
             SdaiSchema schema);

SdaiModel sdaiCreateModelBN (SdaiRep repository,
             SdaiString modelName,
             SdaiString schemaName);
```

Input:

repository:	The identifier of the repository in which the SDAI-model is to be created.
modelName:	The name, unique within the repository, of the new SDAI-model.
schema:	The identifier of the schema upon which the SDAI-model is to be based.
schemaName:	The schema identified by its name instead of its identifier.

Return:

In normal condition:	Identifier of the newly created SDAI-model.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMO_DUP`	SDAI-model duplicate.
`sdaiSD_NDEF`	Schema definition not defined.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.5.1

6.3.2 Create schema instance

The Create Schema Instance function shall create a schema instance based upon the specified schema in the specified repository.

Prototype:

```
SdaiSchemaInstance sdaiCreateSchemaInstance (
             SdaiString schemaInstanceName, SdaiSchema schema,
             SdaiRep repository);

SdaiSchemaInstance sdaiCreateSchemaInstanceBN (
             SdaiString schemaInstanceName, SdaiString schemaName,
             SdaiRep repository);
```

Input:

schemaInstanceName:	The name, unique within the repository, of the schema instance to be created.
schema:	Identifier of the schema upon which the newly created schema instance is to be based.
schemaName:	Name of the schema upon which the newly created schema instance is based.
repository:	Identifier of the repository that shall contain the newly created schema instance.

Return:

In normal condition:	Identifier of the newly created schema instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NEXS`	Repository does not exist.

`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiSI_DUP`	Schema instance duplicate.
`sdaiVT_NVLD`	Name value type is invalid.
`sdaiSD_NDEF`	Schema definition not defined.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.5.2

6.3.3 Close repository

The Close Repository function shall close the specified repository.

Prototype:

```
void sdaiCloseRepository (SdaiRep repository);
```

Input:

repository:	The repository to be closed.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_RW`	Transaction read-write.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.5.3

6.4 Schema instance operations

6.4.1 Delete schema instance

The Delete Schema Instance function shall delete the specified schema instance.

Prototype:

```
void sdaiDeleteSchemaInstance (SdaiSchemaInstance schemaInstance);
void sdaiDeleteSchemaInstanceBN (SdaiString schemaInstanceName,
             SdaiRep repository);
```

Input:

schemaInstance:	Identifier of the schema instance to be deleted.
schemaInstanceName:	The name, unique within the repository, of the schema instance to be deleted.
repository:	Identifier of the repository that contains the schema instance to delete.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiVT_NVLD`	Name value type is invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.6.1

6.4.2 Rename schema instance

The Rename Schema Instance function shall assign a new name to the specified schema instance.

Prototype:

```
void sdaiRenameSchemaInstance (SdaiSchemaInstance schemaInst,
             SdaiString schemaInstName);

void sdaiRenameSchemaInstanceBN (SdaiString schemaInstOldName,
             SdaiRep repository, SdaiString schemaInstName);
```

Input:

schemaInst:	Identifier of the schema instance to be renamed.
schemaInstOldName:	The name, unique within the repository, of the schema instance to be renamed.
repository:	Identifier of the repository that contains the schema instance to rename.
schemaInstName:	New name for the schema instance.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiSI_DUP`	Schema instance duplicate.
`sdaiSI_NEXS`	Schema instance does not exist.

`sdaiRP_NOPN`	Repository is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiVT_NVLD`	Name value type is invalid.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.6.2

6.4.3 Add SDAI-model

The Add Model function shall associate an SDAI-model with the specified schema instance.

Prototype:

```
void sdaiAddModel (SdaiSchemaInstance schemaInstance,
             SdaiModel model);

void sdaiAddModelBN (SdaiSchemaInstance schemaInstance,
             SdaiRep repository, SdaiString modelName);
```

Input:

schemaInstance:	Identifier of the schema instance with which the SDAI-model is to be associated.
model:	Identifier of the SDAI-model that is to be associated with the schema instance.
repository:	The identifier of the repository in which the SDAI-model exists.
modelName:	The name, unique within the repository, of the SDAI-model to be added.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMO_NDEQ`	SDAI-model not domain equivalent.
`sdaiVT_NVLD`	Name value type is invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.6.3

6.4.4 Remove SDAI-model

The Remove Model function shall remove the association of an SDAI-model with a schema instance.

Prototype:

```
void sdaiRemoveModel (SdaiSchemaInstance schemaInstance,
             SdaiModel model);

void sdaiRemoveModelBN (SdaiSchemaInstance schemaInstance,
             SdaiRep repository, SdaiString modelName);
```

Input:

schemaInstance:	Identifier of the schema instance with which the SDAI-model will no longer be associated.
model:	Identifier of the SDAI-model that is to be removed from the schema instance.
repository:	The identifier of the repository in which the SDAI-model exists.
modelName:	The name, unique within the repository, of the SDAI-model to be removed.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMO_NVLD`	SDAI-model invalid.
`sdaiVT_NVLD`	Name value type is invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.6.4

6.4.5 Validate global rule

The Validate Global Rule function shall determine whether the specified global rule is satisfied by the specified schema instance.

Prototype:

```
SdaiLogical sdaiValidateGlobalRule (
             SdaiSchemaInstance schemaInstance,
             SdaiGlobalRule rule, SdaiNPL list);

SdaiLogical sdaiValidateGlobalRuleBN (
             SdaiSchemaInstance schemaInstance,
             SdaiString ruleName, SdaiNPL list);
```

Input:

schemaInstance:	Identifier of the schema instance bounding the validation of the global rule.
rule:	Identifier of the global rule to be validated.
ruleName:	Name of the global rule to be validated.
list:	The identifier of a pre-existing NPL to which the `SdaiInstance` identifiers of those instances are appended that do not conform to the validation.

Return:

In normal condition:	`sdaiTRUE` if the global rule is satisfied; `sdaiFALSE` if the global rule is violated; `sdaiUNKNOWN` if the global rule is not implemented.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRU_NDEF`	Rule not defined.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiED_NVLD`	Entity definition invalid.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22

10.6.5

6.4.6 Validate uniqueness rule

The Validate Uniqueness function shall determine whether the specified uniqueness rule is satisfied by the specified schema instance. In the case of attribute values represented by ADBs, the data value and data type shall be compared, not the ADB identifier.

Prototype:

```
SdaiLogical sdaiValidateUniqueness (
             SdaiSchemaInstance schemaInstance,
             SdaiUniRule uniRule, SdaiNPL list);

SdaiLogical sdaiValidateUniquenessBN (
             SdaiSchemaInstance schemaInstance,
             SdaiString entityName,
             SdaiString uniRuleName, SdaiNPL list);
```

Input:

schemaInstance:	Identifier of the schema instance bounding the validation of the uniqueness rule.
uniRule:	Identifier of the uniqueness rule to be validated.
entityName:	Name of the entity in the schema containing the uniqueness rule.
uniRuleName:	Name of the uniqueness rule in the named entity.
list:	The identifier of a pre-existing NPL to which the `SdaiInstance` identifiers of those instances are appended that do not conform to the validation.

Return:

In normal condition:	`sdaiTRUE` if uniqueness rule is satisfied; `sdaiFALSE` if uniqueness rule is unsatisfied, `sdaiUNKNOWN` if indeterminate.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRU_NDEF`	Rule not defined.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.6.6

6.4.7 Validate instance reference domain

The Validate Reference Domain function shall determine whether all entity-valued attributes in the specified application instance refer to entity instances within SDAI-models in the specified schema instance.

Prototype:

```
SdaiLogical sdaiValidateReferenceDomain (
             SdaiSchemaInstance schemaInstance,
             SdaiAppInstance appInstance, SdaiNPL list);
```

Input:

schemaInstance:	Identifier of the schema instance bounding the validation.
appInstance:	Identifier of the application instance whose references are to be tested.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

Return:

In normal condition:	`sdaiTRUE` if all the reference attributes of the application instance are to entity instances in the correct schema instance; `sdaiFALSE` if any assigned reference is not bounded by the given schema instance; `sdaiUNKNOWN` if any required explicit attribute values are unset that could reference an entity instance.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.6.7

6.4.8 Validate schema instance

The Validate Schema Instance function shall determine whether the schema instance conforms to all constraints specified within the schema upon which the schema instance is based. This operation updates the validation information, including the level of expression evaluation the implementation supports, maintained within the schema instance.

Prototype:

```
SdaiLogical sdaiValidateSchemaInstance (
             SdaiSchemaInstance schemaInstance);
```

Input:

schemaInstance:	Identifier of the schema instance bounding the test.

Return:

In normal condition:	`sdaiTRUE` if all validated constraints are bounded in the schema instance; `sdaiFALSE` if any validated constraint is not bounded by the given schema instance; `sdaiUNKNOWN` if the result cannot be determined.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.5.8

6.4.9 Is validation current

The Is Validation Current function shall determine whether changes have occurred to the specified schema instance or any associated SDAI-model since the most recent invocation of the Validate Schema Instance function.

Prototype:

```
SdaiBoolean sdaiIsValidationCurrent (
              SdaiSchemaInstance schemaInstance);
```

Input:

schemaInstance:	Identifier of the schema instance bounding the test.

Return:

In normal condition:	`sdaiTRUE` if the schema instance validation result is currently set to `sdaiTRUE` and also even valid; `sdaiFALSE` if the schema instance validation result is not set to `sdaiTRUE`, or no longer valid due to modifications at the schema instance contents.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.6.9

6.4.10 Schema instance operations for convenience

6.4.10.1 Get schema definition

The Get Schema function shall return the identifier of the schema definition with the specified name.

Prototype:

```
SdaiSchema sdaiGetSchema (SdaiString schemaName);
```

Input:

schemaName:	The name of the schema to be found.

Return:

In normal condition:	Identifier of the schema definition.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiSD_NDEF`	Schema definition not defined.
`sdaiSY_ERR`	Underlying system error.

6.4.10.2 Get schema instance

The Get Schema Instance function shall return the identifier of the schema instance with the specified name.

Prototype:

```
SdaiSchemaInstance sdaiGetSchemaInstance (
             SdaiString schemaInstanceName, SdaiRep repository);
```

Input:

schemaInstanceName:	Name of the schema instance to be found.
repository:	Repository containing the schema instance.

Return:

In normal condition:	Identifier of the schema instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiSY_ERR`	Underlying system error.

6.5 SDAI-model operations

6.5.1 Delete SDAI-model

The Delete Model function shall delete the specified SDAI-model along with all of the instances it contains.

Prototype:

```
void sdaiDeleteModel (SdaiModel model);

void sdaiDeleteModelBN (SdaiRep repository, SdaiString modelName);
```

Input:

model:	The SDAI-model to delete.
repository:	The identifier of the repository in which the SDAI-model exists.
modelName:	The name, unique within the repository, of the SDAI-model to be deleted.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NEXS`	Transaction does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.

`sdaiRP_NOPN`	Repository is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiVT_NVLD`	Name value type is invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.7.1

6.5.2 Rename SDAI-model

The Rename Model function shall assign a new name to the specified SDAI-model.

Prototype:

```
void sdaiRenameModel (SdaiModel model, SdaiString modelName);

void sdaiRenameModelBN (SdaiRep repository, SdaiString modelOldName,
             SdaiString modelName);
```

Input:

model:	The SDAI-model to rename.
modelName:	The new name for the SDAI-model.
repository:	The identifier of the repository in which the SDAI-model exists.
modelOldName:	The name, unique within the repository, of the SDAI-model to be renamed.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NRW`	Transaction not read-write access mode.
`sdaiRP_NOPN`	Repository is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NEXS`	Transaction does not exist.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMO_DUP`	SDAI-model duplicate.
`sdaiVT_NVLD`	Name value type is invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.7.2

6.5.3 Start SDAI-model access

The Access Model function shall specify a read-write or read-only access mode to the specified SDAI-model.

<u>Prototype:</u>

```
SdaiModel sdaiAccessModel (SdaiModel model, SdaiAccessMode mode);

SdaiModel sdaiAccessModelBN (SdaiRep repository,
              SdaiString modelName,
              SdaiAccessMode mode);
```

<u>Input:</u>

model:	The identifier of the SDAI-model whose access mode is to be assigned.
mode:	The access mode to be assigned to the SDAI-model, can be `sdaiRO` for read-only, `sdaiRW` for read-write.
repository:	The identifier of the repository containing the SDAI-model.
modelName:	The name of the SDAI-model.

<u>Return:</u>

In normal condition:	Identifier of the SDAI-model.
In error condition:	NULL identifier.

<u>Possible error indicators:</u>

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NEXS`	Transaction does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMX_RO`	SDAI-model access read-only.
`sdaiMX_RW`	SDAI-model access read-write.
`sdaiVT_NVLD`	Name value type is invalid.
`sdaiSY_ERR`	Underlying system error.

<u>Original specification in ISO 10303-22:</u>

10.7.3 and 10.7.6

6.5.4 Promote SDAI-model to read-write access

The Promote Model function shall change the access mode of the specified SDAI-model from read-only to read-write.

Prototype:

```
void sdaiPromoteModel (SdaiModel model);
```

Input:

model:	The identifier of the SDAI-model whose access mode is set to be read-write.

Possible error indicators:

sdaiSS_NOPN	Session is not open.
sdaiTR_NRW	Transaction not read-write.
sdaiTR_NEXS	Transaction does not exist.
sdaiTR_NAVL	Transaction currently not available.
sdaiTR_EAB	Transaction ended abnormally.
sdaiRP_NOPN	Repository not open.
sdaiMO_NEXS	SDAI-model does not exist.
sdaiMX_NDEF	SDAI-model access not defined.
sdaiMX_RW	SDAI-model access read-write.
sdaiSY_ERR	Underlying system error.

Original specification in ISO 10303-22:

10.7.4

6.5.5 End SDAI-model access

The End Model Access function shall end access to the specified SDAI-model.

Prototype:

```
void sdaiEndModelAccess (SdaiModel model);
```

Input:

model:	The identifier of the SDAI-model to which access is to be terminated.

Possible error indicators:

sdaiSS_NOPN	Session is not open.
sdaiRP_NOPN	Repository is not open.
sdaiMO_NEXS	SDAI-model does not exist.
sdaiMX_NDEF	SDAI-model access not defined.
sdaiMX_RO	SDAI-model access read-only.
sdaiMX_RW	SDAI-model access read-write.
sdaiTR_RW	Transaction read-write.
sdaiSY_ERR	Underlying system error.

Original specification in ISO 10303-22:

10.7.5 and 10.7.7

6.5.6 Get entity definition

The Get Entity function shall return the entity definition for the specified entity name within the schema definition upon which the specified SDAI-model is based.

Prototype:

```
SdaiEntity sdaiGetEntity (SdaiModel model, SdaiString name);
```

Input:

model:	The SDAI-model based upon the schema definition containing the entity type.
name:	The entity type name.

Return:

In normal condition:	Identifier of the entity definition.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiVT_NVLD`	Name value type is invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.7.8

6.5.7 Create entity instance

The Create Instance function shall create an application instance based upon the specified entity data type in the specified SDAI-model.

Prototype:

```
SdaiAppInstance sdaiCreateInstance (SdaiModel model,
             SdaiEntity entity);

SdaiAppInstance sdaiCreateInstanceBN (SdaiModel model,
             SdaiString entityName);
```

Input:

model:	Identifier of the SDAI-model in which to create the entity instance.
entity:	Identifier of the entity definition upon which the entity instance shall be based.
entityName:	The name of the entity type upon which the entity instance shall be based.

Return:

In normal condition:	Identifier of the newly created entity instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiED_NVLD`	Entity definition invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.7.9

6.5.8 Undo changes

The Undo Changes function shall restore the condition of the contents of the specified SDAI-model to that which existed at the time of the last Access Model with mode set as `sdaiRW` or Save Changes function, whichever occurred most recently.

Prototype:

```
void sdaiUndoChanges (SdaiModel model);
```

Input:

model:	The identifier of the targeted SDAI-model.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.7.10

6.5.9 Save changes

The Save Changes function shall make persistent all changes to the contents of the specified SDAI-model made since the last Access Model with mode set as `sdaiRW`, Save Changes, or Undo Changes function, whichever occurred most recently.

Prototype:

```
void sdaiSaveChanges (SdaiModel model);
```

Input:

model:	The identifier of the target SDAI-model.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.7.11

6.5.10 SDAI-model operations for convenience

6.5.10.1 Create complex entity instance

The Create Complex Instance function shall create a new application instance of the specified type, as determined by a constructed entity type that is made up of the supplied simple entity types, in the specified SDAI-model. For implementations supporting the SDAI data dictionary, this function shall behave as if the Get Complex Entity function was executed to create the entity type in the data dictionary prior to creating the new entity instance.

Prototype:

```
SdaiAppInstance sdaiCreateComplexInstance (SdaiModel model,
             SdaiNPL entityList);

SdaiAppInstance sdaiCreateComplexInstanceBN (SdaiModel model,
             SdaiInteger nameNumber, SdaiString *nameVector);
```

Input:

model:	Identifier of the SDAI-model in which the entity instance will be created.
entityList:	Identifier of a NPL of the `SdaiEntity` entity definitions representing the supplied simple entity types.
nameNumber:	Number of the supplied simple entity type names in the name vector.
nameVector:	Pointer to a vector with the indicated number of `SdaiString` names of the supplied simple entity types.

Return:

In normal condition:	Identifier of the newly created entity instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiED_NVLD`	Entity definition invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

6.5.10.2 Get entity extent

The Get Entity Extent function shall return the identifier of the set instance that is the value of the **entity_extent.instances** (see ISO 10303-22: 8.4.4) attribute where the **entity_extent.definition** is the specified entity type.

Prototype:

```
SdaiSet sdaiGetEntityExtent (SdaiModel model, SdaiEntity entity);

SdaiSet sdaiGetEntityExtentBN (SdaiModel model, SdaiString name);
```

Input:

model:	The identifier of the SDAI-model containing the entity extent.
entity:	The identifier of the entity definition for the entity extent.
name:	The name of the entity type for the entity extent.

Return:

In normal condition:	Identifier of the set containing entity instances in the entity extent.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiSY_ERR`	Underlying system error.

6.6 Scope operations

This subclause describes the functions that address support of the ISO 10303-21 SCOPE construct (see ISO 10303-21: 10.3 and ISO 10303-22: 8.4.5).

6.6.1 Add to scope

The Add To Scope function shall add an application instance into the scope owned by another application instance.

Prototype:

```
void sdaiAddToScope (SdaiAppInstance scopeInstance,
             SdaiAppInstance instance);
```

Input:

scopeInstance:	Identifier of the owning instance into whose scope the owned instance is to be added.
instance:	Identifier of the owned instance to be added to a scope.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiSC_EXS`	Scope exists.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NRW`	SDAI-model access not read-write.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.8.1

6.6.2 Is scope owner

The Is Scope Owner function shall determine whether the specified application instance owns a scope.

<u>Prototype:</u>

```
SdaiLogical sdaiIsScopeOwner (SdaiAppInstance instance);
```

<u>Input:</u>

instance:	Identifier of the instance to be tested for owning a scope.

<u>Return:</u>

In normal condition:	`sdaiTRUE` if instance owns a scope; `sdaiFALSE` if instance does not own a scope.
In error condition:	`sdaiUNKNOWN` if scope is not supported; `sdaiFALSE` otherwise.

<u>Possible error indicators:</u>

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

<u>Original specification in ISO 10303-22:</u>

10.8.2

6.6.3 Get scope

The Get Scope function shall return the identifier of the scope for which the specified application instance is the owner.

<u>Prototype:</u>

```
SdaiScope sdaiGetScope (SdaiAppInstance instance);
```

<u>Input:</u>

instance:	Identifier of the instance whose scope instance is to be returned.

<u>Return:</u>

In normal condition:	Identifier of the scope instance for which the instance is the owner.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiSC_NEXS`	Scope does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.8.3

6.6.4 Remove from scope

The Remove From Scope function shall remove an application instance from the specified scope. If the specified scope is nested within a higher level scope, the application instance shall be added to the next higher level scope. If the application instance is the last member of the scope, the scope shall be deleted.

Prototype:

```
void sdaiRemoveFromScope (SdaiScope scope,
             SdaiAppInstance instance);
```

Input:

scope:	Identifier of the scope that owns the application instance.
instance:	Identifier of the instance to be removed from the scope.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiEI_NAVL`	Entity instance not available.
`sdaiSC_NEXS`	Scope does not exist.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NRW`	SDAI-model access not read-write.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.8.4

6.6.5 Add to export list

The Add To Export List function shall extend the domain of valid references of an application instance by adding it to the export list of a scope. In the case where scopes are nested, the application instance may be added to the export list of more than one scope.

Prototype:

```
void sdaiAddToExportList (SdaiScope scope,
             SdaiAppInstance instance);
```

Input:

scope:	Identifier of the scope with the export list to which the instance shall be added.
instance:	Identifier of the instance to be exported.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiEI_NAVL`	Entity instance not available.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access not read-write.
`sdaiSC_NEXS`	Scope does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.8.5

6.6.6 Remove from export list

The Remove From Export List function shall remove the specified application instance from the export list of the specified scope.

Prototype:

```
void sdaiRemoveFromExportList (SdaiScope scope,
             SdaiAppInstance instance);
```

Input:

scope:	Identifier of the scope owning the export list from which the application instance shall be removed.
instance:	Identifier of the application instance to be removed.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiEI_NAVL`	Entity instance not available.
`sdaiEI_NEXP`	Entity instance not exported.
`sdaiSC_NEXS`	Scope does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access not read-write.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.8.6

6.6.7 Scoped delete

The Scoped Delete function shall delete the application instance owning the specified scope, delete the application instances owned by the specified scope, and delete the specified scope. If any of the application instances owned by the specified scope are themselves scope owners, these scopes are similarly deleted. The scoped deletion of these nested scopes continues until no owned application instance of any nested scope owns a scope.

Prototype:

```
void sdaiScopedDelete (SdaiScope scope);
```

Input:

scope:	Identifier of the scope containing the application instances to be deleted.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiSC_NEXS`	Scope does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access not read-write.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.8.7

6.6.8 Scoped copy in same SDAI-model

The Scoped Copy In Same Model function shall create a copy of all application instances that own and are owned by the specified scope, and populate the copy of the scope based upon the copied application instances all in the same SDAI-model in which the application instance owning the specified scope exists.

Prototype:

```
SdaiScope sdaiScopedCopyInSameModel (SdaiScope scope);
```

Input:

scope:	Identifier of the scope to copy.

Return:

In normal condition:	Identifier of the newly created scope that is a copy of the specified scope.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiSC_NEXS`	Scope does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access not read-write.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.8.8

6.6.9 Scoped copy to other SDAI-model

The Scoped Copy To Other Model function shall create a copy of all application instances that own and are owned by the specified scope, and populate the copy of the scope based upon the copied application instances all in the specified SDAI-model.

Prototype:

```
SdaiScope sdaiScopedCopyToOtherModel (SdaiScope scope,
             SdaiModel model);
```

Input:

scope:	Identifier of the scope to copy.
model:	Identifier of the SDAI-model that is to contain the new application instances and scopes.

Return:

In normal condition:	Identifier of the newly created scope that is a copy of the specified scope.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiSC_NEXS`	Scope does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access not read-write.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMO_NVLD`	SDAI-model invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.8.8

6.6.10 Validate scope reference restrictions

The Validate Scope Reference Restrictions function shall validate the reference restrictions of all instances in the scope of the specified application instance. Any nested scopes shall also be validated.

Prototype:

```
SdaiLogical sdaiValidateScopeReferenceRestrictions (
             SdaiAppInstance instance);
```

Input:

instance:	Identifier of the instance that is the owner of the scope to be validated.

Return:

In normal condition:	`sdaiTRUE` if all restrictions are satisfied; `sdaiFALSE` if reference restrictions are violated; `sdaiUNKNOWN` if any required explicit attribute value was unset that could reference an entity instance.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.8.9

6.6.11 Scope operations for convenience

6.6.11.1 Get owned scope instances

The Get Owned Scope Instances function shall return the aggregate identifier of the owned attribute of the scope (see ISO 10303-22: 8.4.5) owned by the specified application instance.

Prototype:

```
SdaiSet sdaiGetOwnedScopeInstances (SdaiAppInstance appInstance);
```

Input:

appInstance:	Identifier of the application instance owning a scope.

Return:

In normal condition:	Identifier of the set of owned application instances.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEI_NEXS`	Entity instance does not exist. Here also: no scope defined
`sdaiFN_NAVL`	Function not available.

`sdaiSY_ERR`	Underlying system error.

6.6.11.2 Get scope owner

The Get Scope Owner function shall return the identifier of the application instance that is the owner of the scope owning the specified application instance.

Prototype:

```
SdaiAppInstance sdaiGetScopeOwner (SdaiAppInstance appInstance);
```

Input:

appInstance:	Identifier of an owned application instance.

Return:

In normal condition:	Identifier of the owning application instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

6.6.11.3 Get export list

The Get Export List function shall return the aggregate identifier of the **export_list** attribute of the **scope** (see ISO 10303-22: 8.4.5) owned by the specified application instance.

Prototype:

```
SdaiSet sdaiGetExportList (SdaiAppInstance appInstance);
```

Input:

appInstance:	Identifier of the owning application instance.

Return:

In normal condition:	Identifier of the set containing the exported application instances.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

6.7 Type operations

6.7.1 Get complex entity definition

The Get Complex Entity function shall return, and may add to the data dictionary, the constructed **entity_definition** for the entity data type composed of the supplied simple entity types.

Prototype:

```
SdaiEntity sdaiGetComplexEntity (SdaiNPL entityList);

SdaiEntity sdaiGetComplexEntityBN (SdaiString schemaName,
             SdaiInteger nameNumber, SdaiString *nameVector);
```

Input:

entityList:	Identifier of a NPL of `SdaiEntity` entity definitions representing the supplied simple entity types.
schemaName:	Name of the schema to which the entity types belong.
nameNumber:	Number of the supplied simple entity type names in the name vector.
nameVector:	Pointer to a vector with the indicated number of `SdaiString` names of the supplied simple entity types.

Return:

In normal condition:	Identifier of the resulting complex entity definition.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiSD_NDEF`	Schema definition not defined.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiED_NVLD`	Entity definition invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.9.1

6.7.2 Is subtype of

The Is Subtype Of function shall determine whether an entity definition is a subtype of another entity definition.

Prototype:

```
SdaiBoolean sdaiIsSubtypeOf (SdaiEntity subtype,
             SdaiEntity supertype);

SdaiBoolean sdaiIsSubtypeOfBN (SdaiString schemaName,
             SdaiString subName, SdaiString superName);
```

Input:

subtype:	Identifier of an entity definition, to be tested as a subtype.
supertype:	Identifier of an entity definition, to be tested as a supertype.
schemaName:	Name of the schema to which the entity types belong.
subName:	Name of the entity definition, to be tested as a subtype.
superName:	Name of the entity definition, to be tested as a supertype.

Return:

In normal condition:	`sdaiTRUE` if the identified types are the same, or in the assumed relation determined by the application schema(s) and the domain equivalence instructions; `sdaiFALSE` if the relationship does not hold.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiED_NDEQ`	Entity definition not domain equivalent.
`sdaiSD_NDEF`	Schema definition not defined.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.9.2

6.7.3 Is SDAI subtype of

The Is SDAI Subtype Of function shall determine whether an entity definition is a subtype of another entity definition based upon the application schemas and the SDAI parameter data schema found in ISO 10303-22 clause 9.

Prototype:

```
SdaiBoolean sdaiIsSDAISubtypeOf (SdaiEntity subtype,
             SdaiEntity supertype);

SdaiBoolean sdaiIsSDAISubtypeOfBN (SdaiString schemaName,
             SdaiString subName, SdaiString superName);
```

Input:

subtype:	Identifier of an entity definition, to be tested as a subtype.
supertype:	Identifier of an entity definition, to be tested as a supertype.
schemaName:	Name of the schema to which the entity types belong.
subName:	Name of the entity definition, to be tested as a subtype.
superName:	Name of the entity definition, to be tested as a supertype.

Return:

In normal condition:	`sdaiTRUE` if the identified types are the same, or in the assumed relation determined by the SDAI data type schema, the application schema(s) and the domain equivalence instructions; `sdaiFALSE` if the relationship does not hold.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiED_NDEQ`	Entity definition not domain equivalent.
`sdaiSD_NDEF`	Schema definition not defined.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.9.3

6.7.4 Is domain equivalent with

The Is Deq With function shall determine whether an entity data type is domain equivalent with another entity data type.

Prototype:

```
SdaiBoolean sdaiIsDeqWith (
             SdaiEntity entityType1,
             SdaiEntity entityType2);

SdaiBoolean sdaiIsDeqWithBN (
             SdaiString schemaName1, SdaiString entityName1,
             SdaiString schemaName2, SdaiString entityName2);
```

Input:

entityType1:	Identifier of an entity definition, to be tested as domain equivalent.
entityType2:	Identifier of an entity definition, to be tested against as a domain equivalent.
schemaName1:	Name of the schema to which the entity type1 belongs.
schemaName2:	Name of the schema to which the entity type2 belongs.
entityName1:	Name of the entity definition, to be tested as domain equivalent.
entityName2:	Name of the entity definition, to be tested against as domain equivalent.

Return:

In normal condition:	`sdaiTRUE` if the first identified entity type is domain equivalent with the second one; `sdaiFALSE` if the first identified entity type is not domain equivalent with the second one.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiSD_NDEF`	Schema definition not defined.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.9.4

6.7.5 Type operations for convenience

6.7.5.1 Get attribute definition

The Get Attribute Definition function shall return the identifier of the attribute definition from the data dictionary for the specified entity data type.

Prototype:

```
SdaiAttr sdaiGetAttrDefinition (SdaiEntity  entity,
             SdaiString attrName);

SdaiAttr sdaiGetAttrDefinitionBN (SdaiString schemaName,
             SdaiString entityName, SdaiString attrName);
```

Input:

entity:	Identifier of an entity definition.
attrName:	Name of an attribute.
schemaName:	Name of the schema to which the entity type belongs.
entityName:	Name of an entity type.

Return:

In normal condition:	Identifier of the specified attribute.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiSD_NDEF`	Schema definition not defined.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

6.8 Entity instance operations

6.8.1 Get attribute

The Get Attr function shall return, and may convert, the value of an attribute from an entity instance.

Prototype:

```
void* sdaiGetAttr (SdaiInstance instance, SdaiAttr attribute,
             SdaiPrimitiveType valueType, void *value);

void* sdaiGetAttrBN (SdaiInstance instance,
             SdaiString attributeName,
             SdaiPrimitiveType valueType, void *value);
```

Input:

instance:	The entity instance whose attribute is being read.
attribute:	The identifier of the attribute definition of the attribute to be read.

attributeName:	The name of the attribute to be read.
valueType:	The type of the attribute value, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiNUMBER`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, `sdaiINSTANCE`, or `sdaiAGGR`.
value:	Handle matching or convertible to the valueType. If valueType is `sdaiADB`, value shall be the handle of an ADB previously created by a call to one of the ADB create functions described in 6.2.12.1.

Output:

value:	Handle filled with the primitive or identifier value.

Return:

This function shall return the value argument filled with the value read from the attribute. If the value-Type is `sdaiADB`, `sdaiAGGR`, or `sdaiINSTANCE` an identifier shall be returned.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiVA_NSET`	Value not set.
`sdaiVT_NVLD`	Value type invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.1

6.8.2 Test attribute

The Test Attr function shall determine whether the specified explicit attribute a value for the specified entity instance.

Prototype:

```
SdaiBoolean sdaiTestAttr (SdaiInstance instance,
             SdaiAttr attribute);

SdaiBoolean sdaiTestAttrBN (SdaiInstance instance,
             SdaiString attributeName);
```

Input:

instance:	The instance of the entity whose attribute is being tested.
attribute:	An `SdaiAttr` instance from the SDAI dictionary.
attributeName:	The name of the attribute being tested.

Return:

This function shall return `sdaiTRUE` if the attribute has a value or `sdaiFALSE` if the attribute value is not set.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.2

6.8.3 Find entity instance SDAI-model

The Get Instance Model function shall return the identifier of the SDAI-model in which the entity instance exists.

Prototype:

```
SdaiModel sdaiGetInstanceModel (SdaiInstance instance);
```

Input:

instance:	The instance identifier whose SDAI-model is to be found.

Return:

In normal condition:	Identifier of the found SDAI-model.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.

`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.3

6.8.4 Get instance type

The Get Instance Type function shall return the entity type of the specified entity instance.

Prototype:

```
SdaiEntity sdaiGetInstanceType (SdaiInstance instance);
```

Input:

instance:	Identifier of the entity instance whose type is to be returned.

Return:

In normal condition:	Identifier of an entity definition.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.4

6.8.5 Is instance of

The Is Instance Of function shall determine whether the specified entity instance is exactly of, or domain equivalent with, the specified entity type.

Prototype:

```
SdaiBoolean sdaiIsInstanceOf (SdaiInstance instance,
             SdaiEntity entity);

SdaiBoolean sdaiIsInstanceOfBN (SdaiInstance instance,
             SdaiString entityName);
```

Input:

instance:	Identifier of an instance.
entity:	Identifier of an entity definition.
entityName:	Name of an entity data type.

Return:

In normal condition:	`sdaiTRUE` if the instance is of the specified type; `sdaiFALSE` if the instance is not of the specified type.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.5

6.8.6 Is kind of

The Is Kind Of function shall determine whether an entity instance is of the specified entity type or one of its subtypes.

Prototype:

```
SdaiBoolean sdaiIsKindOf (SdaiInstance instance, SdaiEntity entity);

SdaiBoolean sdaiIsKindOfBN (SdaiInstance instance,
             SdaiString entityName);
```

Input:

instance:	Entity instance identifier.
entity:	Entity definition identifier.
entityName:	Entity type name.

Return:

In normal condition:	`sdaiTRUE` if the instance is a kind of the entity type; `sdaiFALSE` if the instance is not a kind of the entity type.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NEXS`	Transaction does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.6

6.8.7 Is SDAI kind of

The Is SDAI Kind Of function shall determine whether or not an entity instance is of the specified entity type, or is of one of its subtypes, based upon the application schemas and the SDAI parameter data schema found in ISO 10303-22 clause 9.

Prototype:

```
SdaiBoolean sdaiIsSDAIKindOf (SdaiInstance instance,
             SdaiEntity entity);

SdaiBoolean sdaiIsSDAIKindOfBN (SdaiInstance instance,
             SdaiString entityName);
```

Input:

instance:	Entity instance identifier.
entity:	Entity definition identifier.
entityName:	Entity type name.

Return:

In normal condition:	`sdaiTRUE` if the instance is an SDAI kind of the entity type; `sdaiFALSE` if the instance is not an SDAI kind of the entity type.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiED_NDEF`	Entity definition not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.7

6.8.8 Find entity instance users

The Find Instance Users function shall return the identifiers of all the entity instances in the defined domain that reference the specified entity instance.

Prototype:

```
SdaiNPL sdaiFindInstanceUsers (SdaiInstance instance,
                 SdaiNPL domain, SdaiNPL resultList);
```

Input:

instance:	Identifier of the entity instance whose users are requested.
domain:	Identifier of a NPL containing the `SdaiSchemaInstance` identifiers of the schema instances that define the domain of the function request.
resultList:	Identifier of the pre-existing NPL to which the `SdaiInstance` instance identifiers of the entity instances referencing the specified entity instance are appended.

Return:

Identifier of the result NPL.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.

`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.8

6.8.9 Find entity instance used in

The Find Instance Used In functions shall return the identifiers of all the entity instances in the defined domain that reference the specified entity instance by the specified attribute.

Prototype:

```
SdaiNPL sdaiFindInstanceUsedIn (SdaiInstance instance,
             SdaiAttr role, SdaiNPL domain, SdaiNPL resultList);

SdaiNPL sdaiFindInstanceUsedInBN (SdaiInstance instance,
             SdaiString roleName, SdaiNPL domain,
             SdaiNPL resultList);
```

Input:

instance:	Identifier of the entity instance whose users are requested.
role:	Identifier of the attribute as the role being requested.
roleName:	A string that contains a fully qualified attribute name as defined in ISO 10303-11: 15.20.
domain:	Identifier of a NPL containing the `SdaiSchemaInstance` identifiers of the schema instances that define the domain of the function request.
resultList:	Identifier of the pre-existing NPL to which the `SdaiInstance` instance identifiers of the entity instances referencing the specified entity instance by the specified attribute are added.

Return:

Identifier of the result NPL.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiEI_NEXS`	Entity instance does not exist.

`sdaiAT_NDEF`	Attribute not defined.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.9

6.8.10 Get attribute value bound

The Get Attr Bound function shall return the current value of the real precision, the string width, or the binary width for the specified attribute of the specified entity instance.

Prototype:

```
SdaiInteger sdaiGetAttrBound (SdaiInstance instance,
             SdaiAttr attribute);

SdaiInteger sdaiGetAttrBoundBN (SdaiInstance instance,
             SdaiString attributeName);
```

Input:

instance:	Identifier of the entity instance whose bound is to be returned.
attribute:	Identifier of a real, string, or binary typed attribute of the instance.
attributeName:	Name of a real, string, or binary typed attribute of the instance.

Return:

The attribute bound value.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiVA_NSET`	Value not set.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiAT_NVLD`	Attribute invalid.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.10

6.8.11 Find instance roles

The Find Instance Roles Of function shall return the identifiers of the attributes of entity instances referencing the specified entity instance in the specified domain.

Prototype:

```
SdaiNPL sdaiFindInstanceRolesOf (SdaiInstance instance,
             SdaiNPL domain, SdaiNPL resultList);
```

Input:

instance:	Identifier of the entity instance whose users are inspected.
domain:	Identifier of a NPL containing the `SdaiSchemaInstance` identifiers of the schema instances that define the domain of the function request.
resultList:	Identifier of the pre-existing NPL to which the `SdaiAttr` identifiers of the entity instances that reference the specified entity instance are appended.

Return:

Identifier of the result NPL.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiSI_NEXS`	Schema instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.11

6.8.12 Find instance data types

The Find Instance Type Of function shall return the identifier of all `SdaiNamedType` data dictionary instances of which the specified entity instance is a member.

Prototype:

```
SdaiNPL sdaiFindInstanceTypeOf (SdaiInstance instance,
             SdaiNPL resultList);
```

Input:

instance:	Identifier of the entity instance whose types are requested.
resultList:	Identifier of the pre-existing NPL to which the `SdaiNamedType` instance identifiers for those dictionary instances are added meeting the specified criteria.

Return:

Identifier of the result NPL.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.10.12

6.8.13 Entity instance operations for convenience

6.8.13.1 Get attributes

The Get Attrs function shall return, and may convert, the values of one or more attributes of the specified entity instance. The behaviour of this function is the same as that of the Get Attr function, except that more than one attribute value may be returned at once.

Prototype:

```
void sdaiGetAttrs (SdaiInstance instance, SdaiInteger numberAttr,
             SdaiAttr attribute, SdaiPrimitiveType valueType,
             void *value, ...);

void sdaiGetAttrsBN (SdaiInstance instance, SdaiInteger numberAttr,
             SdaiString attributeName, SdaiPrimitiveType valueType,
             void *value, ...);
```

Input:

instance:	Identifier of the instance whose attributes are being read.
numberAttr:	Number of attributes to be read and indirectly number of arguments specified in the call.
attribute:	The attribute definition, from the data dictionary, of the attribute to be read.
attributeName:	The name of the attribute to be read.
valueType:	The type of the read attribute, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiNUMBER`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, `sdaiINSTANCE`, or `sdaiAGGR`.
value:	Handle matching or convertible to the valueType. If valueType is `sdaiADB`, value shall be the handle of an ADB previously created by a call to one of the ADB create functions described in 6.2.12.1.

The input parameters attribute or attributeName, valueType and value shall be repeated in the given order as specified by the value of numberAttr.

Output:

value:	These functions shall return the value argument(s) filled with the primitive or identifier attribute value read from the instance. If the valueType is of `sdaiADB`, `sdaiAGGR`, or `sdaiINSTANCE`, an identifier shall be returned.

If the number of arguments does not match the number required for the specified number of attributes, the behaviour of these two functions shall be undefined.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiVT_NVLD`	Value type invalid.
`sdaiVA_NSET`	Value not set.
`sdaiSY_ERR`	Underlying system error.

6.8.13.2 Get all attributes

The Get All Attrs function shall return, and may convert, the values of all explicit attributes of the specified entity instance. An array of `SdaiADB`'s is returned, containing all of the attribute values in the order defined in ISO 10303-21. In the case of an attribute value being unset, an empty ADB shall be returned in the array.

Prototype:

```
SdaiADB *sdaiGetAllAttrs (SdaiInstance instance,
             SdaiInteger *numberAttr);
```

Input:

instance:	Identifier of the instance whose explicit attributes are being read.

Output:

numberAttr:	Number of attribute values returned.

Return:

In normal condition:	This function shall return a C array containing identifiers of internally created `SdaiADB`'s with the value arguments filled with the primitive or identifier attribute values read from the instance. If the attribute is an aggregate or instance, the identifier shall be set in the ADB.
In error condition:	NULL pointer.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiSY_ERR`	Underlying system error.

6.9 Application instance operations

6.9.1 Copy application instance in same SDAI-model

The Near Copy Instance function shall create a new application instance in the same SDAI-model having the same attribute values as the specified entity instance.

Prototype:

```
SdaiAppInstance sdaiNearCopyInstance (SdaiAppInstance instance);
```

Input:

instance:	The entity instance to be copied.

Return:

In normal condition:	Identifier of the newly created instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.1

6.9.2 Copy application instance to other SDAI-model

The Far Copy Instance function shall create a new application instance having the same attribute values as the specified entity instance in the specified SDAI-model instance. The specified SDAI-model shall be associated with a schema instance with which the SDAI-model containing the specified entity instance is associated.

Prototype:

```
SdaiAppInstance sdaiFarCopyInstance (SdaiAppInstance instance,
             SdaiModel model);
```

Input:

instance:	Identifier of the entity instance to be copied.
model:	Identifier of the target SDAI-model.

Return:

In normal condition:	Identifier of the newly created entity instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.

`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiMO_NEXS`	SDAI-model does not exist.
`sdaiMO_NDEQ`	SDAI-model not domain equivalent.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.1

6.9.3 Delete application instance

The Delete Instance function shall delete the specified application instance. Any aggregate instances associated with any aggregate-valued attribute of the deleted application instance shall also be deleted.

Prototype:

```
void sdaiDeleteInstance (SdaiAppInstance instance);
```

Input:

instance:	Identifier of the application instance to be deleted.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.2

6.9.4 Put attribute

The Put Attr function shall set, and may convert, the value of the specified attribute of the specified application instance.

Prototype:

```
void sdaiPutAttr (SdaiAppInstance instance,
             SdaiExplicitAttr attribute,
             SdaiPrimitiveType valueType, ...);

void sdaiPutAttrBN (SdaiAppInstance instance,
             SdaiString attributeName,
             SdaiPrimitiveType valueType, ...);
```

Input:

instance:	Identifier of the application instance.
attribute:	The attribute definition, from the data dictionary, of the attribute to be set.
attributeName:	The name of the attribute to be set.
valueType:	The type of the attribute to be put, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, or `sdaiINSTANCE`.
... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiVT_NVLD`	Value type invalid.
`sdaiAT_NVLD`	Attribute invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.3

6.9.5 Unset attribute value

The Unset Attr function shall restore the state of the specified attribute in the specified application instance such it has no value. A subsequent Test Attr function will return `sdaiFALSE`.

Prototype:

```
void sdaiUnsetAttr (SdaiAppInstance instance,
             SdaiExplicitAttr attribute);

void sdaiUnsetAttrBN (SdaiAppInstance instance,
             SdaiString attributeName);
```

Input:

instance:	The application instance whose attribute is to be unset.
attribute:	An attribute definition from the data dictionary.
attributeName:	The name of the attribute to be unset.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiAT_NVLD`	Attribute invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.4

6.9.6 Create aggregate instance

The Create Aggr function shall create an aggregate instance identifier as the value of the specified attribute of the specified application instance.

Prototype:

```
SdaiAggr sdaiCreateAggr (SdaiAppInstance instance,
             SdaiExplicitAttr attribute);

SdaiAggr sdaiCreateAggrBN (SdaiAppInstance instance,
             SdaiString attributeName);
```

Input:

instance:	Identifier of an application instance.
attribute:	Identifier of the aggregate valued attribute definition of the instance.
attributeName:	Name of the attribute in the entity definition.

Return:

In normal condition:	Identifier of the new aggregate.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiAT_NVLD`	Attribute invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.5

6.9.7 Create aggregate instance ADB

The Create AggrADB function shall create an aggregate instance identifier as the value of the specified attribute of the specified application instance based on the specified ADB.

Prototype:

```
SdaiAggr sdaiCreateAggrADB (SdaiAppInstance instance,
             SdaiExplicitAttr attribute, SdaiADB selaggrInstance);

SdaiAggr sdaiCreateAggrADBBN (SdaiAppInstance instance,
             SdaiString attributeName, SdaiADB selaggrInstance);
```

Input:

instance:	Identifier of an application instance.
attribute:	Identifier of the aggregate valued attribute definition of the instance.
attributeName:	Name of the attribute in the entity definition.
selaggrInstance:	The ADB specifying the type of aggregate to create.

Return:

In normal condition:	Identifier of the new aggregate.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiAT_NVLD`	Attribute invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.5

6.9.8 Get persistent label

The Get Persistent Label function shall return a persistent label for the specified application instance.

Prototype:

```
SdaiString sdaiGetPersistentLabel (SdaiAppInstance instance,
             SdaiString labelBuffer);
```

Input:

instance:	Identifier of the entity instance for which a persistent label is requested.

Output:

labelBuffer:	The pre-allocated buffer filled with the requested persistent label of the maximum length of 256 character signs.

Return:

The function returns the labelBuffer.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NEXS`	Transaction does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEI_NEXS`	Entity instance does not exist.

`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.6

6.9.9 Get session identifier

The Get Session Id function shall return the session identifier of the application instance referenced by the specified persistent label in the specified repository.

Prototype:

```
SdaiAppInstance sdaiGetSessionId (SdaiRep repository,
              SdaiString label);
```

Input:

repository:	Identifier of the repository the label is valid in.
label:	The persistent label of an application instance in the repository.

Return:

In normal condition:	Identifier of the application instance identified by its label.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NEXS`	Repository does not exist.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NEXS`	Transaction does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.7

6.9.10 Get description

The Get Description function shall return a human readable description for the specified application instance.

Prototype:

```
SdaiString sdaiGetDescription (SdaiAppInstance instance,
             SdaiString descriptionBuffer);
```

Input:

instance:	Identifier of the entity instance for which a description is requested.

Output:

descriptionBuffer:	The pre-allocated buffer filled with the requested description of the maximum length of 1024 character signs.

Return:

The function returns the description for the application instance.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NEXS`	Transaction does not exist.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.8

6.9.11 Validate where rule

The Validate Where Rule function shall determine whether the specified where rule is satisfied by the specified application instance.

Prototype:

```
SdaiLogical sdaiValidateWhereRule (SdaiAppInstance instance,
             SdaiWhereRule rule);

SdaiLogical sdaiValidateWhereRuleBN (SdaiAppInstance instance,
             SdaiString ruleName);
```

Input:

instance:	Identifier of an application instance to be validated
rule:	Identifier of the where rule to be evaluated

ruleName:	Label of the where rule to be evaluated.

Return:

In normal condition:	`sdaiTRUE` if rule is satisfied; `sdaiFALSE` if rule is violated; `sdaiUNKNOWN` if indeterminate.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiRU_NDEF`	Rule not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.9

6.9.12 Validate required explicit attributes assigned

The Validate Required Attrs function shall determine whether a value has been set for the mandatory explicit attributes of the specified entity instance.

Prototype:

```
SdaiBoolean sdaiValidateRequiredAttrs (SdaiAppInstance instance,
            SdaiNPL list);
```

Input:

instance:	Identifier of an instance.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

Return:

`sdaiTRUE` if all non-optional attributes of the identified instance have values, or if the instance has no non-optional attributes. `sdaiFALSE` if any non-optional attribute has no value in the instance, or if an error occurs.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.10

6.9.13 Validate inverse attributes

The Validate Inverse Attrs function shall determine whether all EXPRESS INVERSE attribute constraints defined in the specified application instance are satisfied.

Prototype:

```
SdaiBoolean sdaiValidateInverseAttrs (SdaiAppInstance instance,
             SdaiNPL list);
```

Input:

instance:	Identifier of an instance.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

Return:

In normal condition:	`sdaiTRUE` if all inverse attribute constraints are satisfied or if the instance has no inverse attributes; `sdaiFALSE` if any inverse attribute constraint is violated.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiEX_NSUP`	Expression evaluation not supported.

`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.11

6.9.14 Validate explicit attributes references

The Validate Attr Types function shall determine whether all of the entity instances that are values of attributes of the specified application instance are of a valid entity data type for those attributes.

Prototype:

```
SdaiLogical sdaiValidateAttrTypes (SdaiAppInstance instance,
             SdaiNPL list);
```

Input:

instance:	Identifier of an application instance.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

Return:

In normal condition:	`sdaiTRUE` if all entity-valued attributes have instance values of a correct type; `sdaiFALSE` if any entity-valued attribute has an instance value of an incorrect type;`sdaiUNKNOWN` if any required explicit attribute value is unset that could reference an entity instance.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.12

6.9.15 Validate aggregates size

The Validate Aggr Sizes function shall determine whether the aggregate size constraints defined in the data dictionary for the specified application instance are satisfied.

Prototype:

```
SdaiLogical sdaiValidateAggrSizes (SdaiAppInstance instance,
             SdaiNPL list);
```

Input:

instance:	Identifier of the application instance to be evaluated.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

Return:

In normal condition:	`sdaiTRUE` if aggregate size is valid; `sdaiFALSE` if aggregate size is not valid.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiVA_NSET`	Value not set.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.13

6.9.16 Validate aggregates uniqueness

The Validate Aggr Uni function shall determine whether all the aggregate uniqueness constraints defined in the data dictionary for the specified application instance are satisfied. In the case of aggregate members represented by ADBs, the data value and data type shall be compared, not the ADB identifier.

Prototype:

```
SdaiBoolean sdaiValidateAggrUni (SdaiAppInstance instance,
             SdaiNPL list);
```

Input:

instance:	Identifier of an instance.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

Return:

In normal condition:	`sdaiTRUE` if all aggregate uniqueness is satisfied; `sdaiFALSE` if at least one aggregate uniqueness failed.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiVA_NSET`	Value not set.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.14

6.9.17 Validate array not optional

The Validate Array Not Optional function shall determine whether array instances whose array type declaration does not allow optional elements have values at all index positions. The validation is performed for all array instances associated with all attributes of the specified application instance.

Prototype:

```
SdaiBoolean sdaiValidateArrayNotOptional (SdaiAppInstance instance,
             SdaiNPL list);
```

Input:

instance:	Identifier of an entity instance.

list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

<u>Return:</u>

In normal condition:	`sdaiTRUE` if valid; `sdaiFALSE` if invalid.
In error condition:	`sdaiFALSE`.

<u>Possible error indicators:</u>

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiVA_NSET`	Value not set.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

<u>Original specification in ISO 10303-22:</u>

10.11.15

6.9.18 Validate string width

The Validate String Width function shall determine whether all STRING-valued attributes of the specified application instance are of a valid width.

<u>Prototype:</u>

```
SdaiLogical sdaiValidateStringWidth (SdaiAppInstance appInstance,
             SdaiNPL list);
```

<u>Input:</u>

appInstance:	Identifier of the application instance to be validated.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

<u>Return:</u>

In normal condition:	`sdaiTRUE` if the constraint is satisfied; `sdaiFALSE` if the constraint is violated.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.11.16

6.9.19 Validate binary width

The Validate Binary Width function shall determine whether all BINARY-valued attributes of the specified application instance are of a valid width.

Prototype:

```
SdaiLogical sdaiValidateBinaryWidth (SdaiAppInstance appInstance,
            SdaiNPL list);
```

Input:

appInstance:	Identifier of the application instance to be validated.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

Return:

In normal condition:	`sdaiTRUE` if the constraint is satisfied; `sdaiFALSE` if the constraint is violated.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.

`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiVA_NSET`	Value not set.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

<u>Original specification in ISO 10303-22:</u>

10.11.17

6.9.20 Validate real precision

The Validate Real Precision function shall determine whether all REAL-valued attributes of the specified application instance are of the valid minimum precision.

<u>Prototype:</u>

```
SdaiLogical sdaiValidateRealPrecision (SdaiAppInstance appInstance,
             SdaiNPL list);
```

<u>Input:</u>

appInstance:	Identifier of the application instance to be validated.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

<u>Return:</u>

In normal condition:	`sdaiTRUE` if the constraint is satisfied; `sdaiFALSE` if the constraint is violated.
In error condition:	`sdaiUNKNOWN`.

<u>Possible error indicators:</u>

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiVA_NSET`	Value not set.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

<u>Original specification in ISO 10303-22:</u>

10.11.18

6.9.21 Application instance operations for convenience

The functions in this clause are defined to provide for a more efficient and convenient access to more than one attributes by a single function call.

6.9.21.1 Put attributes

The Put Attrs function shall set, and may convert, the values of one or more explicit attributes of the specified application instance.

Prototype:

```
void sdaiPutAttrs (SdaiAppInstance appInstance,
             SdaiInteger numberAttr, SdaiExplicitAttr attribute,
             SdaiPrimitiveType valueType, ...);

void sdaiPutAttrsBN (SdaiAppInstance appInstance,
             SdaiInteger numberAttr, SdaiString attributeName,
             SdaiPrimitiveType valueType, ...);
```

Input:

appInstance:	Identifier of the application instance whose attributes are to be set.
numberAttr:	Number of explicit attributes to be set and indirectly number of arguments specified in the call.
attribute:	The attribute definition of the explicit attribute to be set.
attributeName:	The name of the explicit attribute to be set.
valueType:	Type of the attribute to be set, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, `sdaiINSTANCE`, or `sdaiAGGR`.
... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

The input parameters attribute or attributeName, valueType and the value parameter shall repeated in the given order as specified by the value of numberAttr.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiAT_NVLD`	Attribute invalid.

`sdaiVT_NVLD`	Value type invalid.
`sdaiSY_ERR`	Underlying system error.

6.9.21.2 Put all attributes

The Put All Attrs function shall set the values of all explicit attributes of the specified application instance.

Prototype:

```
void sdaiPutAllAttrs (SdaiAppInstance appInstance,
             SdaiInteger numberAttr, SdaiADB *values);
```

Input:

appInstance:	Identifier of the application instance whose explicit attributes are being set.
numberAttr:	Number of attribute values are being provided, determines the length of the values array.
values:	An array of C structures `SdaiADB` containing the types and values of the explicit attributes in the order defined in ISO 10303-21.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiEI_NEXS`	Instance does not exist.
`sdaiAT_NVLD`	Attribute invalid.
`sdaiAT_NDEF`	Attribute not defined.
`sdaiVT_NVLD`	Value type invalid.
`sdaiSY_ERR`	Underlying system error.

6.10 Entity instance aggregate operations

6.10.1 Get member count

The Get Member Count function shall return the number of elements contained in the specified aggregate. If the aggregate is an array, the size of the array is returned.

Prototype:

```
SdaiInteger sdaiGetMemberCount (SdaiAggr aggregate);
```

Input:

aggregate:	Identifier of an aggregate.

Return:

In normal condition:	The number of elements or the size of the array.
In error condition:	-1.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.1

6.10.2 Is member

The Is Member function shall determine whether the specified primitive or instance value is contained in the aggregate. In the case of aggregate members represented by ADBs, both the data value and data type shall be compared.

Prototype:

```
SdaiBoolean sdaiIsMember (SdaiAggr aggregate,
             SdaiPrimitiveType valueType, ...);
```

Input:

aggregate:	Identifier of the aggregate.
valueType:	One of the following types: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, `sdaiINSTANCE`, and `sdaiAGGR`.
... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Return:

In normal condition:	`sdaiTRUE` if the specified value is a member; `sdaiFALSE` if it is not a member.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiVT_NVLD`	Value type invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.2

6.10.3 Create iterator

The Create Iterator function shall create an iterator associated with the specified aggregate instance. The iterator shall be positioned as if the Beginning function had been executed such that so that no member of the aggregate is referenced as the current member.

Prototype:

```
SdaiIterator sdaiCreateIterator (SdaiAggr aggregate);
```

Input:

aggregate:	Identifier of an aggregate with which an iterator is to be associated.

Return:

In normal condition:	Identifier of an iterator.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.3

6.10.4 Delete iterator

The Delete Iterator function shall delete the specified iterator.

Prototype:

```
void sdaiDeleteIterator (SdaiIterator iterator);
```

Input:

iterator:	Identifier of the iterator to be deleted.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.4

6.10.5 Beginning

The Beginning function shall position the iterator at the beginning of its associated aggregate instance such that there is no current member.

Prototype:

```
void sdaiBeginning (SdaiIterator iterator);
```

Input:

iterator:	Identifier of an iterator.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiAI_NSET`	Aggregate instance is empty.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.5

6.10.6 Next

The Next function shall position the iterator to the succeeding member of the associated aggregate instance.

Prototype:

```
SdaiBoolean sdaiNext (SdaiIterator iterator);
```

Input:

iterator:	Identifier of the iterator.

Return:

In normal condition:	`sdaiTRUE` if there is a new current element; `sdaiFALSE` if there is no new current element.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiAI_NSET`	Aggregate instance is empty.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.6

6.10.7 Get current member

The Get Aggr By Iterator function shall return, and may convert, the value of the current member referenced by the specified iterator.

Prototype:

```
void* sdaiGetAggrByIterator (SdaiIterator iterator,
             SdaiPrimitiveType valueType, void *value);
```

Input:

iterator:	Identifier of the iterator identifying the member of interest.
valueType:	Type of the aggregate element to be read, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiNUMBER`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, `sdaiINSTANCE`, or `sdaiAGGR`.
value:	Handle matching or convertible to the valueType. If valueType is `sdaiADB`, value shall be the handle of an ADB previously created by a call to one of the ADB create functions described in 6.2.12.1.

Output:

value:	Handle filled with the primitive or identifier value read from the aggregate.

Return:

This function shall return the value argument filled with the primitive or identifier value read from the aggregate member. If the valueType is `sdaiADB`, `sdaiAGGR`, or `sdaiINSTANCE`, an identifier shall be returned. If the member has no value, none shall be returned and an error shall be set.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiAI_NSET`	Aggregate instance is empty.
`sdaiIR_NSET`	Iterator has no current value.
`sdaiVT_NVLD`	Value type invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.7

6.10.8 Get value bound by iterator

The Get Aggr Element Bound By Itr function shall return the current value of the real precision, the string width, or the binary width for the current member referenced by the specified iterator.

Prototype:

```
SdaiInteger sdaiGetAggrElementBoundByItr (SdaiIterator iterator);
```

Input:

iterator:	Identifier of iterator specifying the aggregate element whose bound value is to be returned.

Return:

The aggregate element bound value.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiVA_NSET`	Value not set.
`sdaiVT_NVLD`	Value type invalid.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.8

6.10.9 Get lower bound

The Get Lower Bound function shall return the current value of the lower bound, or index, of the specified aggregate instance.

Prototype:

```
SdaiInteger sdaiGetLowerBound (SdaiAggr aggregate);
```

Input:

aggregate:	Identifier of the aggregate instance whose lower bound value is to be returned.

Return:

The aggregate lower bound value.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.

`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiVA_NSET`	Value not set.
`sdaiVT_NVLD`	Value type invalid.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.9

6.10.10 Get upper bound

The Get Upper Bound function shall return the current value of the upper bound, or index, of the specified aggregate instance.

Prototype:

```
SdaiInteger sdaiGetUpperBound (SdaiAggr aggregate);
```

Input:

aggregate:	Identifier of the aggregate instance whose upper bound value is to be returned.

Return:

The aggregate upper bound value.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiVA_NSET`	Value not set.
`sdaiVT_NVLD`	Value type invalid.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.12.10

6.11 Application instance aggregate operations

6.11.1 Create aggregate instance as current member

The Create Nested Aggr By Itr function shall create an aggregate instance replacing the current member of the aggregate instance referenced by the specified iterator. In the case where the type of the aggregate to create is a SELECT TYPE and ambiguous, the type shall be specified on input using an ADB. The function shall set the value of the ADB with the identifier of the newly created aggregate instance.

Prototype:

```
SdaiAggr sdaiCreateNestedAggrByItr (SdaiIterator current);

SdaiAggr sdaiCreateNestedAggrByItrADB (SdaiIterator current,
              SdaiADB selaggrInstance);
```

Input:

current:	The iterator referencing the current member of the aggregate.
selaggrInstance:	The ADB specifying the type of aggregate to create.

Return:

In normal condition:	The identifier of the newly created aggregate instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiIR_NSET`	Iterator has no current value.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiAB_NEXS`	ADB does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.13.1

6.11.2 Put current member

The Put Aggr By Iterator function shall replace, and may convert, the value of the current member of an aggregate referenced by the specified iterator.

Prototype:

```
void sdaiPutAggrByIterator (SdaiIterator iterator,
             SdaiPrimitiveType valueType, ...);
```

Input:

iterator:	The iterator referencing the aggregate member to replace.
valueType:	Type of the value to be set, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, or `sdaiINSTANCE`.
... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiVT_NVLD`	Value type invalid.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiAI_NSET`	Aggregate instance is empty.
`sdaiIR_NSET`	Iterator has no current value.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.13.2

6.11.3 Remove current member

The Remove By Itr function shall remove the current member of an aggregate instance, that is not an array, referenced by the specified iterator. After executing the function, the iterator position shall be set as if the Next function had been invoked before the member was removed. If the removed member was an aggregate instance, it along with any nested aggregate instances shall be deleted.

Prototype:

```
void sdaiRemoveByIterator (SdaiIterator iterator);
```

Input:

iterator	Identifier of an iterator, not associated with an array.

Possible error indicators:

sdaiSS_NOPN	Session is not open.
sdaiTR_NAVL	Transaction currently not available.
sdaiTR_EAB	Transaction ended abnormally.
sdaiTR_NRW	Transaction not read-write.
sdaiRP_NOPN	Repository is not open.
sdaiMX_NRW	SDAI-model access is not read-write.
sdaiVT_NVLD	Value type invalid.
sdaiAI_NEXS	Aggregate instance does not exist.
sdaiAI_NSET	Aggregate instance is empty.
sdaiAI_NVLD	Aggregate instance invalid.
sdaiIR_NEXS	Iterator does not exist.
sdaiIR_NSET	Iterator has no current value.
sdaiSY_ERR	Underlying system error.

Original specification in ISO 10303-22:

10.13.3

6.12 Application instance unordered collection operations

6.12.1 Add unordered

The Add function shall add a new member to an unordered aggregate instance.

Prototype:

```
void sdaiAdd (SdaiUnorderedAggr unorderedAggr,
             SdaiPrimitiveType valueType, ...);
```

Input:

unorderedAggr:	An identifier of an unordered aggregate instance.
valueType:	Type of the value to be set, one of the following argument values: sdaiINTEGER, sdaiREAL, sdaiBOOLEAN, sdaiLOGICAL, sdaiSTRING, sdaiBINARY, sdaiENUM, sdaiADB, or sdaiINSTANCE.
... :	Value of SdaiInteger, SdaiReal, SdaiBoolean, SdaiLogical type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiVT_NVLD`	Value type invalid.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.14.1

6.12.2 Create aggregate instance unordered

The Create Nested Aggr function shall create an aggregate instance as a member of an unordered aggregate instance. In the case where the type of the aggregate to create is a SELECT TYPE and ambiguous, the type shall be specified on input using an ADB. The function shall set the value of the ADB with the identifier of the newly created aggregate instance.

Prototype:

```
SdaiAggr sdaiCreateNestedAggr (SdaiUnorderedAggr aggregate);

SdaiAggr sdaiCreateNestedAggrADB (SdaiUnorderedAggr aggregate,
              SdaiADB selaggrInstance);
```

Input:

aggregate:	Identifier of an unordered aggregate instance.
selaggrInstance:	The ADB specifying the type of aggregate to create.

Return:

In normal condition:	The identifier of the newly created aggregate instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.

`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.14.2

6.12.3 Remove unordered

The Remove function shall remove one occurrence of the specified value from the specified unordered aggregate instance. If the removed element is an aggregate instance, it along with any nested aggregate instances shall be deleted.

Prototype:

```
void sdaiRemove (SdaiUnorderedAggr unorderedAggr,
             SdaiPrimitiveType valueType, ...);
```

Input:

unorderedAggr:	An identifier of an unordered aggregate instance.
valueType:	Type of the value to be removed, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, `sdaiAGGR`, or `sdaiINSTANCE`.
... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiVT_NVLD`	Value type invalid.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiVA_NEXS`	Value does not exist.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.14.3

6.13 Entity instance ordered collection operations

6.13.1 Get by index

The Get Aggr By Index function shall get, and may convert, the value of the aggregate member referenced by the specified index position.

Prototype:

```
void* sdaiGetAggrByIndex (SdaiOrderedAggr aggregate,
             SdaiAggrIndex index, SdaiPrimitiveType valueType,
             void *value);
```

Input:

aggregate:	Identifier of an ordered aggregate instance.
index:	Position in the aggregate of the value to be returned.
valueType:	Type of the value to be read, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiNUMBER`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, `sdaiINSTANCE`, or `sdaiAGGR`.
value:	Handle matching or convertible to the valueType. If valueType is `sdaiADB`, value shall be the handle of an ADB previously created by a call to one of the ADB create functions described in 6.2.12.1.

Output:

value:	Handle filled with the primitive or identifier value read from the aggregate.

Return:

This function shall return the value argument filled with the primitive or identifier value read from the aggregate member. If the valueType is `sdaiADB`, `sdaiAGGR`, or `sdaiINSTANCE`, an identifier shall be returned. If the member has no value, none shall be returned and an error shall be set.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiVT_NVLD`	Value type invalid.
`sdaiVA_NSET`	Value not set.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIX_NVLD`	Index invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.15.1

6.13.2 End

The End function shall position the specified iterator at the end of the ordered aggregate instance members such that there is no current member.

Prototype:

```
void sdaiEnd (SdaiIterator iterator);
```

Input:

iterator:	Iterator identifier for an ordered aggregate instance.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiAI_NSET`	Aggregate instance is empty.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.15.2

6.13.3 Previous

The Previous function shall position the specified iterator so that the preceding member of its subject ordered aggregate instance shall become the current member. If the iterator is at the end of the aggregate, the last member shall become the current member. If the iterator is at the beginning of the aggregate no repositioning shall occur. If the iterator references the first member of the aggregate, the iterator shall be set at the beginning so there is no current member.

Prototype:

```
SdaiBoolean sdaiPrevious (SdaiIterator iterator);
```

Input:

iterator:	Identifier of an iterator of an ordered aggregate instance.

Return:

In normal condition:	`sdaiTRUE` if there is a member at the new current position; `sdaiFALSE` if there is no member at the new current position.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiAI_NSET`	Aggregate instance is empty.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.15.3

6.13.4 Get value bound by index

The Get Aggr Element Bound By Index function shall return the current value of the real precision, the string width, or the binary width of the aggregate element at the specified index position in the specified ordered aggregate instance.

Prototype:

```
SdaiInteger sdaiGetAggrElementBoundByIndex (
             SdaiOrderedAggr aggregate, SdaiAggrIndex index);
```

Input:

aggregate:	Identifier of aggregate instance containing the aggregate element whose bound value is to be returned.
index:	Position specifying the aggregate element whose bound value is to be returned.

Return:

The aggregate element bound value.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.

sdaiMX_NDEF	SDAI-model access not defined.
sdaiIX_NVLD	Index invalid.
sdaiAI_NEXS	Aggregate instance does not exist.
sdaiVA_NSET	Value not set.
sdaiVT_NVLD	Value type invalid.
sdaiEX_NSUP	Expression evaluation not supported.
sdaiFN_NAVL	Function not available.
sdaiSY_ERR	Underlying system error.

Original specification in ISO 10303-22:

10.15.4

6.14 Application instance ordered collection operations

6.14.1 Put by index

The Put Aggr By Index function shall replace, and may convert, the value of the member of the specified ordered aggregate instance referenced by the specified index.

Prototype:

```
void sdaiPutAggrByIndex (SdaiOrderedAggr aggregate,
             SdaiAggrIndex index, SdaiPrimitiveType valueType, ...);
```

Input:

aggregate:	Identifier of an ordered aggregate instance.
index:	Position in the aggregate of the value to be set.
valueType:	Type of the value to be set, one of the following argument values: sdaiINTEGER, sdaiREAL, sdaiBOOLEAN, sdaiLOGICAL, sdaiSTRING, sdaiBINARY, sdaiENUM, sdaiADB, or sdaiINSTANCE.
... :	Value of SdaiInteger, SdaiReal, SdaiBoolean, SdaiLogical type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Possible error indicators:

sdaiSS_NOPN	Session is not open.
sdaiTR_NAVL	Transaction currently not available.
sdaiTR_EAB	Transaction ended abnormally.
sdaiTR_NRW	Transaction not read-write.
sdaiRP_NOPN	Repository is not open.
sdaiMX_NRW	SDAI-model access is not read-write.
sdaiVT_NVLD	Value type invalid.
sdaiAI_NEXS	Aggregate instance does not exist.
sdaiAI_NVLD	Aggregate instance invalid.
sdaiIX_NVLD	Index invalid.

`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.16.1

6.14.2 Create aggregate instance by index

The Create Nested Aggr By Index function shall create an aggregate instance and replaces the existing member of the specified ordered aggregate instance referenced by the specified index. If the replaced member was an aggregate instance, it along with any nested aggregate instances shall be deleted. In the case where the type of the aggregate to create is a SELECT TYPE and ambiguous, the type shall be specified on input using an ADB. The function shall set the value of the ADB with the identifier of the newly created aggregate instance.

Prototype:

```
SdaiAggr sdaiCreateNestedAggrByIndex (SdaiOrderedAggr aggregate,
             SdaiAggrIndex index);

SdaiAggr sdaiCreateNestedAggrByIndexADB (SdaiOrderedAggr aggregate,
             SdaiAggrIndex index, SdaiADB selaggrInstance);
```

Input:

aggregate:	Identifier of a nested ordered aggregate instance.
index:	The index at where the new aggregate will be set.
selaggrInstance:	The ADB specifying the type of aggregate to create.

Return:

In normal condition:	The identifier of the newly created aggregate instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction access not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiIX_NVLD`	Index invalid.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.16.2

6.15 Entity instance array operations

6.15.1 Test by index

The Test Array By Index function shall test whether the member of the specified array referenced by the specified index position has a value.

Prototype:

```
SdaiBoolean sdaiTestArrayByIndex (SdaiArray array,
             SdaiAggrIndex index);
```

Input:

array:	Identifier of an array aggregate instance.
index:	Position in the array aggregate to be tested.

Return:

In normal condition:	`sdaiTRUE` if the array member is assigned a value; `sdaiFALSE` if the member value is undefined.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiIX_NVLD`	Index invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.17.1

6.15.2 Test current member

The Test Array By Itr function shall test whether the member of the array referenced by the specified iterator has a value.

Prototype:

```
SdaiBoolean sdaiTestArrayByItr (SdaiIterator iterator);
```

Input:

iterator:	Identifier of an iterator of an array position to be tested.

Return:

In normal condition:	`sdaiTRUE` if the identified array member is assigned a value; `sdaiFALSE` if the member value is undefined.
In error condition:	`sdaiFALSE`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiIR_NSET`	Current member is not defined.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.17.2

6.15.3 Get lower index

The Get Lower Index function shall return the value of the lower index of the specified array instance when it was created.

Prototype:

```
SdaiInteger sdaiGetLowerIndex (SdaiArray array);
```

Input:

array:	Identifier of the array instance whose lower index value is to be returned.

Return:

The array lower index value.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.

`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.17.3

6.15.4 Get upper index

The Get Upper Index function shall return the value of the upper index of the specified array instance when it was created.

Prototype:

```
SdaiInteger sdaiGetUpperIndex (SdaiArray array);
```

Input:

array:	Identifier of the array instance whose upper index value is to be returned.

Return:

The array upper index value.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.17.4

6.16 Application instance array operations

6.16.1 Unset value by index

The Unset Array By Index function shall restore the unset (not assigned a value) status of the member of the specified array at the specified index position. If the array member value was previously an aggregate instance, it along with any nested aggregate instances shall be deleted.

Prototype:

```
void sdaiUnsetArrayByIndex (SdaiArray array, SdaiAggrIndex index);
```

Input:

array:	Array that contains the value to be unset.
index:	Index identifying the value in the array to be unset.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiIX_NVLD`	Index invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.18.1

6.16.2 Unset value current member

The Unset Array By Itr function shall restore the unset (not assigned a value) status of a member at the position identified by the iterator in the array associated with the iterator. Only iterators associated with array aggregate instances may be specified as argument to this function. If the array member value was previously an aggregate instance, it along with any nested aggregate instances shall be deleted.

Prototype:

```
void sdaiUnsetArrayByItr (SdaiIterator iterator);
```

Input:

iterator:	Identifier of an iterator associated with an array aggregate instance.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiIR_NSET`	Current member is not defined.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.18.2

6.16.3 Reindex array

The Reindex Array function shall resize the specified array instance setting the lower, or upper index, or both, based upon the current population of the application schema.

Prototype:

```
void sdaiReindexArray (SdaiArray array);
```

Input:

array:	Identifier of the array instance to be reindexed.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.18.3

6.16.4 Reset array index

The Reset Array Index function shall resize the specified array instance setting the lower and upper index with the specified values.

Prototype:

```
void sdaiResetArrayIndex (SdaiArray array, SdaiAggrIndex lower,
             SdaiAggrIndex upper);
```

Input:

array:	Identifier of the array instance to be reindexed.
lower:	The value for the new lower index.
upper:	The value for the new upper index.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiVA_NVLD`	Value invalid.
`sdaiVT_NVLD`	Value type invalid.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.18.4

6.17 Application instance list operations

6.17.1 Add before current member

The Insert Before function shall insert a new member with the specified value before the current member referenced by the specified iterator whose subject is a list instance. The current member referenced by the iterator shall be unchanged by this function.

Prototype:

```
void sdaiInsertBefore (SdaiIterator iterator,
             SdaiPrimitiveType valueType, ...);
```

Input:

iterator:	Identifier of an iterator whose subject is a list aggregate instance.
valueType:	Type of the value to be set, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, or `sdaiINSTANCE`.
... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiVT_NVLD`	Value type invalid.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiIR_NSET`	Iterator has no current value.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.19.1

6.17.2 Add after current member

The Insert After function shall insert a new member with the specified value after the current member referenced by the specified iterator whose subject is a list instance. The current member referenced by the iterator shall be unchanged by this function.

Prototype:

```
void sdaiInsertAfter (SdaiIterator iterator,
             SdaiPrimitiveType valueType, ...);
```

Input:

iterator:	Identifier of an iterator whose subject is a list aggregate instance.
valueType:	Type of the value to be set, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, or `sdaiINSTANCE`.

... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiVT_NVLD`	Value type invalid.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiIR_NSET`	Iterator has no current value.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.19.2

6.17.3 Add by index

The Insert By Index function shall add a new member with the specified value to the specified list instance. The position of the new member within the list instance shall be specified by the index.

Prototype:

```
void sdaiInsertByIndex (SdaiList list, SdaiAggrIndex index,
             SdaiPrimitiveType valueType, ...);
```

Input:

list:	Identifier of a list instance.
index:	Index identifying the position for the new list element.
valueType:	Type of the value to be set, one of the following argument values: `sdaiINTEGER`, `sdaiREAL`, `sdaiBOOLEAN`, `sdaiLOGICAL`, `sdaiSTRING`, `sdaiBINARY`, `sdaiENUM`, `sdaiADB`, or `sdaiINSTANCE`.
... :	Value of `SdaiInteger`, `SdaiReal`, `SdaiBoolean`, `SdaiLogical` type, or handle matching or convertible to the valueType, and given as a function parameter with the specified C late binding type.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiRP_NOPN`	Repository is not open.

`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiIX_NVLD`	Index invalid.
`sdaiVT_NVLD`	Value type invalid.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.19.3

6.17.4 Create aggregate instance before current member

The Insert Nested Aggr Before function shall create an aggregate instance as a member of a list instance. The newly created aggregate shall be inserted into the list instance before the member referenced by the specified iterator. In the case where the type of the aggregate to create is a SELECT TYPE and ambiguous, the type shall be specified on input using an ADB. The function shall set value of the ADB with the identifier of the newly created aggregate instance.

Prototype:

```
SdaiAggr sdaiInsertNestedAggrBefore (SdaiIterator iterator);

SdaiAggr sdaiInsertNestedAggrBeforeADB (SdaiIterator iterator,
             SdaiADB selaggrInstance);
```

Input:

iterator:	Iterator connected with a nested list aggregate instance.
selaggrInstance:	The ADB specifying the type of aggregate to create.

Return:

In normal condition:	The identifier of the newly created aggregate instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction access not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.

`sdaiIR_NEXS`	Iterator does not exist.
`sdaiIR_NSET`	Iterator not set.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiSY_ERR`	Underlying system error.

<u>Original specification in ISO 10303-22:</u>

10.19.4

6.17.5 Create aggregate instance after current member

The Insert Nested Aggr After function shall create an aggregate instance as a member of a list instance. The newly created aggregate shall be inserted into the list instance after the member referenced by the specified iterator. In the case where the type of the aggregate to create is a SELECT TYPE and ambiguous, the type shall be specified on input using an ADB. The function shall set the value of the ADB with the identifier of the newly created aggregate instance.

<u>Prototype:</u>

```
SdaiAggr sdaiInsertNestedAggrAfter (SdaiIterator iterator);

SdaiAggr sdaiInsertNestedAggrAfterADB (SdaiIterator iterator,
              SdaiADB selaggrInstance);
```

<u>Input:</u>

iterator:	Iterator connected with a nested list aggregate instance.
selaggrInstance:	The ADB specifying the type of aggregate to create.

<u>Return:</u>

In normal condition:	The identifier of the newly created aggregate instance.
In error condition:	NULL identifier.

<u>Possible error indicators:</u>

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction access not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiIR_NEXS`	Iterator does not exist.
`sdaiIR_NSET`	Iterator not set.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.

`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.19.5

6.17.6 Add aggregate instance by index

The Insert Nested Aggr By Index function shall create an aggregate instance as a member of the specified list instance. The newly created aggregate shall be inserted into the list instance at the position referenced by the specified index. In the case where the type of the aggregate to create is a SELECT TYPE and ambiguous, the type shall be specified on input using an ADB. The function shall set the value of the ADB with the identifier of the newly created aggregate instance.

Prototype:

```
SdaiAggr sdaiInsertNestedAggrByIndex (SdaiList list,
             SdaiAggrIndex index);

SdaiAggr sdaiInsertNestedAggrByIndexADB (SdaiList list,
             SdaiAggrIndex index, SdaiADB selaggrInstance);
```

Input:

list:	Identifier of a nested list aggregate instance.
index:	The index at where the new aggregate instance will be inserted.
selaggrInstance:	The ADB specifying the type of aggregate to create.

Return:

In normal condition:	The identifier of the newly created aggregate instance.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction access not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiIX_NVLD`	Index invalid.
`sdaiVA_NSET`	Value not set.
`sdaiEX_NSUP`	Expression evaluation not supported.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.19.6

6.17.7 Remove by index

The Remove By Index function shall remove the member of the specified list referenced by the specified index. If the removed member is an aggregate instance, it along with any nested aggregate instances shall be deleted.

Prototype:

```
void sdaiRemoveByIndex (SdaiList list, SdaiAggrIndex index);
```

Input:

list:	Identifier of the list to be modified.
index:	The list position of the member to be removed.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiIX_NVLD`	Index invalid.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiAI_NVLD`	Aggregate instance invalid.
`sdaiSY_ERR`	Underlying system error.

Original specification in ISO 10303-22:

10.19.7

6.18 C late binding specific SELECT TYPE operations

In the case where the defined types applicable to an attribute value or aggregate member resulting from one or more EXPRESS SELECT TYPEs are ambiguous, an ADB is used as the mechanism to implement an instance of a **select_value** as specified in ISO 10103-22: 9.4.8. The ADB is used to specify both the value and applicable defined types. Implementations of this part of ISO 10303 shall provide the specified Type Path functions to establish and retrieve the **data_type** of the **select_value**. The **data_type** shall be passed across the interface as a vector that supports one or more `SdaiString` as its members. The names of the applicable defined data types are set, on input by the calling application and on output by the implementation, as the vector members.

6.18.1 Put ADB type path

The Put ADB Type Path function shall set the **data_type** value of the specified ADB with the EXPRESS TYPE names in the specified vector. The application is responsible for allocation of memory for the input vector. After the function executes the application is responsible for freeing the allocated memory.

Prototype:

```
void sdaiPutADBTypePath (SdaiADB block, SdaiInteger typeNameNumber,
             SdaiString *typeNameVector);
```

Input:

block:	Identifier of the ADB.
typeNameNumber:	The count of the EXPRESS TYPE names in the vector.
typeNameVector:	Pointer to the vector of EXPRESS TYPE names.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiTR_NRW`	Transaction not read-write.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NRW`	SDAI-model access is not read-write.
`sdaiVA_NVLD`	Value invalid.
`sdaiVT_NVLD`	Value type invalid.
`sdaiAB_NEXS`	ADB does not exist.
`sdaiSY_ERR`	Underlying system error.

6.18.2 Get ADB type path

The Get ADB Type Path function shall return a pointer to a vector of strings representing the type in the specified ADB. The specified ADB is one containing type path information that has been filled by a function reading an attribute or aggregate member value. The implementation is responsible for allocating the memory for the vector referenced by the returned pointer.The contents of this vector shall be valid until the next call of this function. The implementation shall free the memory allocated for the vector upon the application deleting the ADB or closing the SDAI session.

Prototype:

```
SdaiString *sdaiGetADBTypePath (SdaiADB block,
             SdaiInteger *typeNameNumber);
```

Input:

block:	Identifier of an ADB.

Output:

typeNameNumber:	Count of the number of types returned.

Return:

In normal condition:	Pointer to a vector with the indicated number of `SdaiString` type that are the EXPRESS type names comprising the type path information.
In error condition:	NULL identifier.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiVT_NVLD`	Value type invalid.
`sdaiAB_NEXS`	ADB does not exist.
`sdaiSY_ERR`	Underlying system error.

6.18.3 Validate type path

The Validate Type Path function determines whether all attribute and aggregate member values of the specified application instance have defined type names specified as the type path information that are allowable based upon the entity data type upon which the application instance is based.

Prototype:

```
SdaiLogical sdaiValidateTypePath (SdaiAppInstance appInstance,
             SdaiNPL list);
```

Input:

appInstance:	Identifier of the application instance to be validated.
list:	The identifier of a pre-existing NPL to which the `SdaiAttr` identifiers of those attributes are appended that do not conform to the validation.

Return:

In normal condition:	`sdaiTRUE` if the constraint is satisfied; `sdaiFALSE` if the constraint is violated.
In error condition:	`sdaiUNKNOWN`.

Possible error indicators:

`sdaiSS_NOPN`	Session is not open.
`sdaiTR_NAVL`	Transaction currently not available.
`sdaiTR_EAB`	Transaction ended abnormally.
`sdaiRP_NOPN`	Repository is not open.
`sdaiMX_NDEF`	SDAI-model access not defined.

`sdaiEI_NEXS`	Entity instance does not exist.
`sdaiAI_NEXS`	Aggregate instance does not exist.
`sdaiFN_NAVL`	Function not available.
`sdaiSY_ERR`	Underlying system error.

Annex A
(normative)

Information object registration

To provide for unambiguous identification of an information object in an open system, the object identifier

```
{ iso standard 10303 part(24) version(1) }
```

is assigned to this part of ISO 10303. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

Annex B
(informative)

The C late binding header include file <sdai.h>

```
/******************************************************************/
/*     This header file <sdai.h> is related to ISO 10303-24         */
/******************************************************************/

#ifndef _SDAI_
#define _SDAI_

/*** Constant declarations *****************************************/

/* LOGICAL and BOOLEAN value elements: */

#define sdaiFALSE   0
#define sdaiTRUE    1
#define sdaiUNKNOWN 2

/* BINARY value elements: */

#define sdaiBIT0 '0'
#define sdaiBIT1 '1'

/* EXPRESS built-in constants: */

#define sdaiE  2.71828182845904513536028747l3
#define sdaiPI 3.14159265358979323846264338 32
```

```
/* SDAI C late binding error codes: ********************************/

#define sdaiNO_ERR        0   /* No error */
#define sdaiSS_OPN       10   /* Session open */
#define sdaiSS_NAVL      20   /* SDAI not available */
#define sdaiSS_NOPN      30   /* Session is not open */
#define sdaiRP_NEXS      40   /* Repository does not exist */
#define sdaiRP_NAVL      50   /* Repository not available */
#define sdaiRP_OPN       60   /* Repository open */
#define sdaiRP_NOPN      70   /* Repository is not open */
#define sdaiTR_EAB       80   /* Transaction ended abnormally */
#define sdaiTR_EXS       90   /* Transaction exists */
#define sdaiTR_NAVL     100   /* Transaction currently not available */
#define sdaiTR_RW       110   /* Transaction read-write */
#define sdaiTR_NRW      120   /* Transaction not read-write */
#define sdaiTR_NEXS     130   /* Transaction does not exist */
#define sdaiMO_NDEQ     140   /* SDAI-model not domain equivalent */
#define sdaiMO_NEXS     150   /* SDAI-model does not exist */
#define sdaiMO_NVLD     160   /* SDAI-model invalid */
#define sdaiMO_DUP      170   /* SDAI-model duplicate */
#define sdaiMX_NRW      180   /* SDAI-model access not read-write */
```

```
#define sdaiMX_NDEF      190   /* SDAI-model access not defined */
#define sdaiMX_RW        200   /* SDAI-model access read-write */
#define sdaiMX_RO        210   /* SDAI-model access read-only */
#define sdaiSD_NDEF      220   /* Schema definition not defined */
#define sdaiED_NDEF      230   /* Entity definition not defined */
#define sdaiED_NDEQ      240   /* Entity definition not domain equiv.*/
#define sdaiED_NVLD      250   /* Entity definition invalid */
#define sdaiRU_NDEF      260   /* Rule not defined */
#define sdaiEX_NSUP      270   /* Expression evaluation not supported */
#define sdaiAT_NVLD      280   /* Attribute invalid */
#define sdaiAT_NDEF      290   /* Attribute not defined */
#define sdaiSI_DUP       300   /* Schema instance duplicate */
#define sdaiSI_NEXS      310   /* Schema instance does not exist */
#define sdaiEI_NEXS      320   /* Entity instance does not exist */
#define sdaiEI_NAVL      330   /* Entity instance not available */
#define sdaiEI_NVLD      340   /* Entity instance invalid */
#define sdaiEI_NEXP      350   /* Entity instance not exported */
#define sdaiSC_NEXS      360   /* Scope does not exist */
#define sdaiSC_EXS       370   /* Scope exists */
#define sdaiAI_NEXS      380   /* Aggregate instance does not exist */
#define sdaiAI_NVLD      390   /* Aggregate instance invalid */
#define sdaiAI_NSET      400   /* Aggregate instance is empty */
#define sdaiVA_NVLD      410   /* Value invalid */
#define sdaiVA_NEXS      420   /* Value does not exist */
#define sdaiVA_NSET      430   /* Value not set */
#define sdaiVT_NVLD      440   /* Value type invalid */
#define sdaiIR_NEXS      450   /* Iterator does not exist */
#define sdaiIR_NSET      460   /* Current member is not defined */
#define sdaiIX_NVLD      470   /* Index invalid */
#define sdaiER_NSET      480   /* Event recording not set */
#define sdaiOP_NVLD      490   /* Operator invalid */
#define sdaiFN_NAVL      500   /* Function not available */
#define sdaiAB_NEXS      800   /* ADB does not exist */
#define sdaiSY_ERR      1000   /* Underlying system error */

/*** Type declarations ********************************************/

typedef unsigned char       SdaiBit;

/* C late binding simple data types: */
typedef long                SdaiInteger;
typedef double              SdaiReal;
typedef SdaiReal            SdaiNumber;
typedef int                 SdaiBoolean;
typedef int                 SdaiLogical;
typedef char               *SdaiString;
typedef SdaiBit            *SdaiBinary;

/* enumeration data type: */
typedef char               *SdaiEnum;

/* entity instance identifier type: */
typedef SdaiId              SdaiInstance;
```

```
/* aggregate data types: */
typedef SdaiInstance        SdaiAggr;
typedef SdaiAggr            SdaiOrderedAggr;
typedef SdaiAggr            SdaiUnorderedAggr;
typedef SdaiOrderedAggr     SdaiArray;
typedef SdaiOrderedAggr     SdaiList;
typedef SdaiUnorderedAggr   SdaiSet;
typedef SdaiUnorderedAggr   SdaiBag;

/* attribute type data type: */
typedef enum {
      sdaiADB,      sdaiAGGR,     sdaiBINARY,   sdaiBOOLEAN,
      sdaiENUM,     sdaiINSTANCE, sdaiINTEGER,  sdaiLOGICAL,
      sdaiNOTYPE,   sdaiNUMBER,   sdaiREAL,     sdaiSTRING
} SdaiPrimitiveType;

/* SDAI instance identifier types: */
typedef SdaiInstance        SdaiAppInstance;
typedef SdaiInstance        SdaiModel;
typedef SdaiInstance        SdaiRep;
typedef SdaiInstance        SdaiSession;
typedef SdaiInstance        SdaiAttr;
typedef SdaiAttr            SdaiExplicitAttr;
typedef SdaiInstance        SdaiNamedType;
typedef SdaiNamedType       SdaiEntity;
typedef SdaiNamedType       SdaiDefinedType;
typedef SdaiInstance        SdaiWhereRule;
typedef SdaiInstance        SdaiUniRule;
typedef SdaiInstance        SdaiGlobalRule;
typedef SdaiInstance        SdaiSchema;
typedef SdaiInstance        SdaiScope;
typedef SdaiInstance        SdaiSchemaInstance;
typedef SdaiInstance        SdaiTrx;

/* SDAI iterator identifier type: */
typedef SdaiItrId           SdaiIterator;

/* Non-persistent list data type: */
typedef SdaiList            SdaiNPL;

/* Query source data type: */
typedef enum {
      sdaiAGGR, sdaiMODEL, sdaiREP, sdaiSCHEMAINSTANCE
} SdaiQuerySourceType;

/* access mode data type: */
typedef enum {
      sdaiRO, sdaiRW
} SdaiAccessMode;

/* C late binding ADB identifier type: */
typedef SdaiADBId           SdaiADB;

/* aggregate index data type: */
```

```
typedef SdaiIndexId         SdaiAggrIndex;

/* error code data type: */
typedef SdaiErrorId         SdaiErrorCode;

/* error handler data type: */
typedef void              (*SdaiErrorHandler)(SdaiErrorCode);

/* transaction commit mode data type: */
typedef enum    {
      sdaiABORT, sdaiCOMMIT
} SdaiCommitMode;

/* NULL identifier data type: */
typedef SdaiId              SdaiNullId;

/* the ANSI-C style function prototypes of the SDAI operations */
SdaiSession sdaiOpenSession (void);
int sdaiLogicalCompare (SdaiLogical value1, SdaiLogical value2);
SdaiErrorCode sdaiErrorQuery (void);
void sdaiSetErrorHandler (SdaiErrorHandler function);
SdaiErrorHandler sdaiRestoreErrorHandler (void);
SdaiBoolean sdaiIsEqual (SdaiInstance instance1,
               SdaiInstance instance2);

void  sdaiRecordEvent (SdaiSession session, SdaiString functionName,
               SdaiErrorCode error, SdaiString description);
SdaiBoolean sdaiSetEventRecording (SdaiSession session,
               SdaiBoolean setRec);
void sdaiCloseSession (SdaiSession session);
SdaiRep sdaiOpenRepository (SdaiSession session,
               SdaiRep repository);
SdaiRep sdaiOpenRepositoryBN (SdaiSession session,
               SdaiString repositoryName);
SdaiTrx sdaiStartTrx (SdaiSession session, SdaiAccessMode mode);
void sdaiBreakTrx (SdaiTrx transaction, SdaiCommitMode mode);
void sdaiEndTrx (SdaiTrx transaction, SdaiCommitMode mode);
SdaiNPL sdaiCreateNPL (void);
void sdaiDeleteNPL (SdaiNPL list);
SdaiInteger sdaiQuery (SdaiQuerySourceType sourceType,
               SdaiString criteria, SdaiInstance instance,
               SdaiNPL result, ...);
SdaiLogical sdaiIsRecordingOn (SdaiSession session);
SdaiADB sdaiCreateADB (SdaiPrimitiveType valueType, ...);
SdaiADB sdaiCreateEmptyADB (void);
void *sdaiGetADBValue (SdaiADB block,
               SdaiPrimitiveType valueType, void *value);
void sdaiPutADBValue (SdaiADB block,
               SdaiPrimitiveType valueType, ...);
SdaiPrimitiveType sdaiGetADBType (SdaiADB block);
void sdaiUnsetADB (SdaiADB block);
void sdaiDeleteADB (SdaiADB block);
```

```
SdaiModel sdaiCreateModel (SdaiRep repository,
              SdaiString modelName, SdaiSchema schema);
SdaiModel sdaiCreateModelBN (SdaiRep repository,
              SdaiString modelName, SdaiString schemaName);
SdaiSchemaInstance sdaiCreateSchemaInstance (
              SdaiString schemaInstanceName, SdaiSchema schema,
              SdaiRep repository);
SdaiSchemaInstance sdaiCreateSchemaInstanceBN (
              SdaiString schemaInstanceName, SdaiString schemaName,
              SdaiRep repository);
void  sdaiCloseRepository (SdaiRep repository);

void sdaiDeleteSchemaInstance (SdaiSchemaInstance schemaInstance);
void sdaiDeleteSchemaInstanceBN (SdaiString schemaInstanceName,
              SdaiRep repository);
void sdaiRenameSchemaInstance (SdaiSchemaInstance schemaInstance,
              SdaiString schemaInstanceName);
void sdaiRenameSchemaInstanceBN (SdaiString schemaInstanceOldName,
              SdaiRep repository, SdaiString schemaInstanceName);
void  sdaiAddModel (SdaiSchemaInstance schemaInstance,
              SdaiModel model);
void  sdaiAddModelBN (SdaiSchemaInstance schemaInstance,
              SdaiRep repository, SdaiString modelName);
void sdaiRemoveModel (SdaiSchemaInstance schemaInstance, SdaiModel model);
void sdaiRemoveModelBN (SdaiSchemaInstance schemaInstance,
              SdaiRep repository, SdaiString modelName);
SdaiLogical sdaiValidateGlobalRule (SdaiSchemaInstance schemaInstance,
              SdaiGlobalRule rule, SdaiNPL list);
SdaiLogical sdaiValidateGlobalRuleBN (
              SdaiSchemaInstance schemaInstance,
              SdaiString ruleName, SdaiNPL list);
SdaiLogical sdaiValidateUniqueness (SdaiSchemaInstance schemaInstance,
              SdaiUniRule uniRule, SdaiNPL list);
SdaiLogical sdaiValidateUniquenessBN (
              SdaiSchemaInstance schemaInstance, SdaiString entityName,
              SdaiString uniRuleName, SdaiNPL list);
SdaiLogical sdaiValidateReferenceDomain (
              SdaiSchemaInstance schemaInstance,
              SdaiAppInstance appInstance, SdaiNPL list);
SdaiLogical sdaiValidateSchemaInstance (
              SdaiSchemaInstance schemaInstance);
SdaiBoolean sdaiIsValidationCurrent (
              SdaiSchemaInstance schemaInstance);
SdaiSchema sdaiGetSchema (SdaiString schemaName);
SdaiSchemaInstance  sdaiGetSchemaInstance (
              SdaiString schemaInstanceName, SdaiRep repository);

void sdaiDeleteModel (SdaiModel model);
void sdaiDeleteModelBN (SdaiRep repository, SdaiString modelName);
void sdaiRenameModel (SdaiModel model, SdaiString modelName);
void sdaiRenameModelBN (SdaiRep repository, SdaiString modelOldName,
              SdaiString modelName);
SdaiModel sdaiAccessModel (SdaiModel model, SdaiAccessMode mode);
SdaiModel sdaiAccessModelBN (SdaiRep repository, SdaiString modelName,
              SdaiAccessMode mode);
void sdaiPromoteModel (SdaiModel model);
void sdaiEndModelAccess (SdaiModel model);
SdaiEntity sdaiGetEntity (SdaiModel model, SdaiString name);
SdaiAppInstance sdaiCreateInstance (SdaiModel model,
              SdaiEntity entity);
SdaiAppInstance sdaiCreateInstanceBN (SdaiModel model,
              SdaiString entityName);
void sdaiUndoChanges (SdaiModel model);
void sdaiSaveChanges (SdaiModel model);
SdaiAppInstance sdaiCreateComplexInstance (SdaiModel model,
              SdaiNPL entityList);
SdaiAppInstance sdaiCreateComplexInstanceBN (SdaiModel model,
              SdaiInteger nameNumber, SdaiString *nameVector);
SdaiSet sdaiGetEntityExtent (SdaiModel model, SdaiEntity entity);
SdaiSet sdaiGetEntityExtentBN (SdaiModel model, SdaiString name);

void sdaiAddToScope (SdaiAppInstance scopeInstance,
              SdaiAppInstance instance);
SdaiLogical sdaiIsScopeOwner (SdaiAppInstance instance);
SdaiScope sdaiGetScope (SdaiAppInstance instance);
void sdaiRemoveFromScope (SdaiScope scope, SdaiAppInstance instance);
void sdaiAddToExportList (SdaiScope scope, SdaiAppInstance instance);
void sdaiRemoveFromExportList (SdaiScope scope,
              SdaiAppInstance instance);
void sdaiScopedDelete (SdaiScope scope);
SdaiScope sdaiScopedCopyInSameModel (SdaiScope scope);
SdaiScope sdaiScopedCopyToOtherModel (SdaiScope scope,
              SdaiModel model);
SdaiLogical sdaiValidateScopeReferenceRestrictions (
              SdaiAppInstance instance);
SdaiSet sdaiGetOwnedScopeInstances (SdaiAppInstance appInstance);
SdaiAppInstance sdaiGetScopeOwner (SdaiAppInstance appInstance);
SdaiSet sdaiGetExportList (SdaiAppInstance appInstance);

SdaiEntity sdaiGetComplexEntity (SdaiNPL entityList);
SdaiEntity sdaiGetComplexEntityBN (SdaiString schemaName,
              SdaiInteger nameNumber, SdaiString *nameVector);
SdaiBoolean sdaiIsSubtypeOf (SdaiEntity subtype, SdaiEntity supertype);
SdaiBoolean sdaiIsSubtypeOfBN (SdaiString schemaName,
              SdaiString subName, SdaiString superName);
SdaiBoolean sdaiIsSDAISubtypeOf (SdaiEntity subtype,
              SdaiEntity supertype);
SdaiBoolean sdaiIsSDAISubtypeOfBN (SdaiString schemaName,
              SdaiString subName, SdaiString superName);
SdaiBoolean sdaiIsDeqWith (SdaiEntity entityType1,
              SdaiEntity entityType2);
SdaiBoolean sdaiIsDeqWithBN (
              SdaiString schemaName1, SdaiString entityName1,
              SdaiString schemaName2, SdaiString entityName2);
SdaiAttr sdaiGetAttrDefinition (SdaiEntity  entity,
              SdaiString attrName);
```

```
SdaiAttr sdaiGetAttrDefinitionBN (SdaiString schemaName,
               SdaiString entityName, SdaiString attrName);
void* sdaiGetAttr (SdaiInstance instance, SdaiAttr attribute,
               SdaiPrimitiveType valueType, void *value);
void* sdaiGetAttrBN (SdaiInstance instance, SdaiString attributeName,
               SdaiPrimitiveType valueType, void *value);
SdaiBoolean sdaiTestAttr (SdaiInstance instance, SdaiAttr attribute);
SdaiBoolean sdaiTestAttrBN (SdaiInstance instance,
               SdaiString attributeName);
SdaiModel  sdaiGetInstanceModel (SdaiInstance instance);
SdaiEntity sdaiGetInstanceType (SdaiInstance instance);
SdaiBoolean sdaiIsInstanceOf (SdaiInstance instance,
               SdaiEntity entity);
SdaiBoolean sdaiIsInstanceOfBN (SdaiInstance instance,
               SdaiString entityName);
SdaiBoolean sdaiIsKindOf (SdaiInstance instance, SdaiEntity entity);
SdaiBoolean sdaiIsKindOfBN (SdaiInstance instance,
               SdaiString entityName);
SdaiBoolean sdaiIsSDAIKindOf (SdaiInstance instance,
               SdaiEntity entity);
SdaiBoolean sdaiIsSDAIKindOfBN (SdaiInstance instance,
               SdaiString entityName);
SdaiNPL sdaiFindInstanceUsers (SdaiInstance instance,
               SdaiNPL domain, SdaiNPL resultList);
SdaiNPL sdaiFindInstanceUsedIn (SdaiInstance instance, SdaiAttr role,
               SdaiNPL domain, SdaiNPL resultList);
SdaiNPL sdaiFindInstanceUsedInBN (SdaiInstance instance,
               SdaiString roleName, SdaiNPL domain,
               SdaiNPL resultList);
SdaiInteger sdaiGetAttrBound (SdaiInstance instance,
               SdaiAttr attribute);
SdaiInteger sdaiGetAttrBoundBN (SdaiInstance instance,
               SdaiString attributeName);
SdaiNPL sdaiFindInstanceRolesOf (SdaiInstance instance,
               SdaiNPL domain, SdaiNPL resultList);
SdaiNPL sdaiFindInstanceTypeOf (SdaiInstance instance,
               SdaiNPL resultList);
void  sdaiGetAttrs (SdaiInstance instance, SdaiInteger numberAttr,
               SdaiAttr attribute, SdaiPrimitiveType valueType,
               void *value, ...);
void  sdaiGetAttrsBN (SdaiInstance instance, SdaiInteger numberAttr,
               SdaiString attributeName, SdaiPrimitiveType valueType,
               void *value, ...);
SdaiADB *sdaiGetAllAttrs (SdaiInstance instance,
               SdaiInteger *numberAttr);

SdaiAppInstance sdaiNearCopyInstance (SdaiAppInstance instance);
SdaiAppInstance sdaiFarCopyInstance (SdaiAppInstance instance,
               SdaiModel model);
void  sdaiDeleteInstance (SdaiAppInstance instance);
void  sdaiPutAttr (SdaiAppInstance instance,
               SdaiExplicitAttr attribute,
               SdaiPrimitiveType valueType, ...);
```

```
void  sdaiPutAttrBN (SdaiAppInstance instance,
              SdaiString attributeName,
              SdaiPrimitiveType valueType, ...);
void  sdaiUnsetAttr (SdaiAppInstance instance,
              SdaiExplicitAttr attribute);
void  sdaiUnsetAttrBN (SdaiAppInstance instance,
              SdaiString attributeName);
SdaiAggr sdaiCreateAggr (SdaiAppInstance instance,
              SdaiExplicitAttr attribute);
SdaiAggr sdaiCreateAggrBN (SdaiAppInstance instance,
              SdaiString attributeName);
SdaiAggr sdaiCreateAggrADB (SdaiAppInstance instance,
              SdaiExplicitAttr attribute, SdaiADB selaggrInstance);
SdaiAggr sdaiCreateAggrADBBN (SdaiAppInstance instance,
              SdaiString attributeName, SdaiADB selaggrInstance);
SdaiString sdaiGetPersistentLabel (SdaiAppInstance instance,
              SdaiString labelBuffer);
SdaiAppInstance sdaiGetSessionId (SdaiRep repository, SdaiString label);
SdaiString sdaiGetDescription (SdaiAppInstance instance,
              SdaiString descriptionBuffer);
SdaiLogical sdaiValidateWhereRule (SdaiAppInstance instance,
              SdaiWhereRule rule);
SdaiLogical sdaiValidateWhereRuleBN (SdaiAppInstance instance,
              SdaiString ruleName);
SdaiBoolean sdaiValidateRequiredAttrs (SdaiAppInstance instance,
              SdaiNPL list);
SdaiBoolean sdaiValidateInverseAttrs (SdaiAppInstance instance,
              SdaiNPL list);
SdaiLogical sdaiValidateAttrTypes (SdaiAppInstance instance,
              SdaiNPL list);
SdaiLogical sdaiValidateAggrSizes (SdaiAppInstance instance,
              SdaiNPL list);
SdaiBoolean sdaiValidateAggrUni (SdaiAppInstance instance,
              SdaiNPL list);
SdaiBoolean sdaiValidateArrayNotOptional (SdaiAppInstance instance,
              SdaiNPL list);
SdaiLogical sdaiValidateStringWidth (SdaiAppInstance appInstance,
              SdaiNPL list);
SdaiLogical sdaiValidateBinaryWidth (SdaiAppInstance appInstance,
              SdaiNPL list);
SdaiLogical sdaiValidateRealPrecision (SdaiAppInstance appInstance,
              SdaiNPL list);
void sdaiPutAttrs (SdaiAppInstance appInstance,
              SdaiInteger numberAttr, SdaiExplicitAttr attribute,
              SdaiPrimitiveType valueType, ...);
void sdaiPutAttrsBN (SdaiAppInstance appInstance,
              SdaiInteger numberAttr, SdaiString attributeName,
              SdaiPrimitiveType valueType, ...);
void sdaiPutAllAttrs (SdaiAppInstance appInstance,
              SdaiInteger numberAttr, SdaiADB *values);

SdaiInteger sdaiGetMemberCount (SdaiAggr aggregate);
SdaiBoolean sdaiIsMember (SdaiAggr aggregate,
              SdaiPrimitiveType valueType, ...);
```

```
SdaiIterator sdaiCreateIterator (SdaiAggr aggregate);
void sdaiDeleteIterator (SdaiIterator iterator);
void sdaiBeginning (SdaiIterator iterator);
SdaiBoolean sdaiNext (SdaiIterator iterator);
void* sdaiGetAggrByIterator (SdaiIterator iterator,
              SdaiPrimitiveType valueType, void *value);
SdaiInteger sdaiGetAggrElementBoundByItr (SdaiIterator iterator);
SdaiInteger sdaiGetLowerBound (SdaiAggr aggregate);
SdaiInteger sdaiGetUpperBound (SdaiAggr aggregate);

SdaiAggr sdaiCreateNestedAggrByItr (SdaiIterator current);
SdaiAggr sdaiCreateNestedAggrByItrADB (SdaiIterator current,
              SdaiADB selaggrInstance);
void sdaiPutAggrByIterator (SdaiIterator iterator,
              SdaiPrimitiveType valueType, ...);
void sdaiRemoveByIterator (SdaiIterator iterator);
void sdaiAdd (SdaiUnorderedAggr unorderedAggr,
              SdaiPrimitiveType valueType, ...);
SdaiAggr sdaiCreateNestedAggr (SdaiUnorderedAggr aggregate);
SdaiAggr sdaiCreateNestedAggrADB (SdaiUnorderedAggr aggregate,
              SdaiADB selaggrInstance);
void sdaiRemove (SdaiUnorderedAggr unorderedAggr,
              SdaiPrimitiveType valueType, ...);
void* sdaiGetAggrByIndex (SdaiOrderedAggr aggregate,
              SdaiAggrIndex index,
              SdaiPrimitiveType valueType, void *value);
void sdaiEnd (SdaiIterator iterator);
SdaiBoolean sdaiPrevious (SdaiIterator iterator);
SdaiInteger sdaiGetAggrElementBoundByIndex (
              SdaiOrderedAggr aggregate,
              SdaiAggrIndex index);

void sdaiPutAggrByIndex (SdaiOrderedAggr aggregate,
              SdaiAggrIndex index,
              SdaiPrimitiveType valueType, ...);
SdaiAggr sdaiCreateNestedAggrByIndex (SdaiOrderedAggr aggregate,
              SdaiAggrIndex index);
SdaiAggr sdaiCreateNestedAggrByIndexADB (SdaiOrderedAggr aggregate,
              SdaiAggrIndex index, SdaiADB selaggrInstance);
SdaiBoolean sdaiTestArrayByIndex (SdaiArray array,
              SdaiAggrIndex index);
SdaiBoolean sdaiTestArrayByItr (SdaiIterator iterator);
SdaiInteger sdaiGetLowerIndex (SdaiArray array);
SdaiInteger sdaiGetUpperIndex (SdaiArray array);
void sdaiUnsetArrayByIndex (SdaiArray array, SdaiAggrIndex index);
void sdaiUnsetArrayByItr (SdaiIterator iterator);
void sdaiReindexArray (SdaiArray array);
void sdaiResetArrayIndex (SdaiArray array, SdaiAggrIndex lower,
              SdaiAggrIndex upper);

void sdaiInsertBefore (SdaiIterator iterator,
              SdaiPrimitiveType valueType, ...);
void sdaiInsertAfter (SdaiIterator iterator,
              SdaiPrimitiveType valueType, ...);
```

```
void sdaiInsertByIndex (SdaiList list, SdaiAggrIndex index,
               SdaiPrimitiveType valueType, ...);
SdaiAggr sdaiInsertNestedAggrBefore (SdaiIterator iterator);
SdaiAggr sdaiInsertNestedAggrBeforeADB (SdaiIterator iterator,
               SdaiADB selaggrInstance);
SdaiAggr sdaiInsertNestedAggrAfter (SdaiIterator iterator);
SdaiAggr sdaiInsertNestedAggrAfterADB (SdaiIterator iterator,
               SdaiADB selaggrInstance);
SdaiAggr sdaiInsertNestedAggrByIndex (SdaiList list,
               SdaiAggrIndex index);
SdaiAggr sdaiInsertNestedAggrByIndexADB (SdaiList list,
               SdaiAggrIndex index, SdaiADB selaggrInstance);
void sdaiRemoveByIndex (SdaiList list, SdaiAggrIndex index);

void sdaiPutADBTypePath (SdaiADB block, SdaiInteger typeNameNumber,
               SdaiString *typeNameVector );
SdaiString *sdaiGetADBTypePath (SdaiADB block,
               SdaiInteger *typeNameNumber);
SdaiLogical sdaiValidateTypePath (SdaiAppInstance appInstance,
               SdaiNPL list);

#endif /* _SDAI_ */
```

Index

Access mode 13, 21, 42
Access model 42, 45
Add 101
Add model 33
Add to export list 51
Add to scope 48
Aggregate 85–87, 91–120
ANSI C 1, 15
Array 86, 112–114
Attribute
  explicit 12, 37, 55, 62, 72, 75, 82, 84
  inverse 83
Attribute data block 3, 4, 9, 14, 26–29
Beginning 94
Binary width 69, 88, 96, 106
Break transaction 22
Close repository 31
Close session 17, 20
Copy
  far 74
  near 73
  scoped 53
Create ADB 26
Create aggregate 77
  nested 102
  nested, by index 108
  nested, by iterator 99, 117, 118
  using ADB 78
Create complex instance 46
Create instance 44
Create iterator 93
Create model 29
Create NPL 23
Create schema instance 30
Delete ADB 29
Delete instance 75
  scoped 52
Delete iterator 94
Delete model 40
Delete NPL 23
Delete schema instance 31
Domain equivalent 4, 59, 64
End 105
End model access 43
End transaction 22
Entity definition 44, 57
Error

codes 7, 14, 15
handling 14, 17–18
Event recording 19–20, 25
Export list
add to 51
get 56
remove from 51
EXPRESS
aggregate types 10
BINARY 9, 69, 88, 96, 106
BOOLEAN 8
built-in constants 7
ENTITY 10
ENUMERATION 9
INTEGER 8
INVERSE 83
LOGICAL 9
NUMBER 8
REAL 8, 69, 96, 106
ROLESOF 70
SELECT 9
STRING 9, 69, 96, 106
TYPEOF 70
USEDIN 68
Far copy instance 74
Find instance roles of 70
Find instance type of 70
Find instance usedin 68
Find instance users 67
Function 15
Function prototype 4, 15
Get ADB type 28
Get ADB type path 121
Get ADB value 26
Get aggr by index 104
Get aggr by iterator 95
Get aggr element bound by index 106
Get aggr element bound by iterator 96
Get all attributes 72
Get attribute 61
Get attribute bound 69
Get attribute definition 60
Get attributes 71
Get complex entity 46, 57
Get description 80
Get entity 44
Get entity extent 47
Get export list 56
Get instance model 63

Get instance type 64
Get lower bound 97
Get lower index 110
Get member count 91
Get owned scope instances 55
Get persistent label 79
Get schema 39
Get schema instance 39
Get scope 49
Get scope owner 56
Get session identifier 80
Get upper bound 98
Get upper index 111
Global rule 34
Handle C type 4, 10, 13, 14
Header file 6, 125
Implementation class 5
Implementation specific handle
    SdaiADBId 14
    SdaiId 10
    SdaiIndexId 14
    SdaiItrId 13
Information object registration 124
Insert after 115
Insert before 114
Insert by index 116
Insert nested aggr after 118
Insert nested aggr before 117
Insert nested aggr by index 119
Instance identifiers 11
Is deq with 59
Is equal 18
Is instance of 64
Is kind of 65
Is member 92
Is recording on 25
Is scope owner 49
Is SDAI kind of 66
Is SDAI subtype of 59
Is subtype of 58
Is validation current 38
ISO 10303-11 . . . . . 2, 7, 68
ISO 10303-22 . . . . . 1, 2, 5, 15, 47, 55, 59, 66
Iterator 13, 93–97, 99–101, 105, 109, 112, 117
Logical compare 16
Lower bound 97
Lower index 110, 113, 114
Macro 6
Memory management 5

Near copy instance 73
Next 95, 100
Non-persistent list 13
    create 23
    delete 23
Open repository 20
Open session 16, 17, 20
Previous 105
Primitive 10
Promote model 42
Put ADB type path 121
Put ADB value 27
Put aggr by index 107
Put aggr by iterator 100
Put all attributes 91
Put attribute 75
Put attributes 90
Query 13, 24
Real precision 69, 89, 96, 106
Record event 19
References 84
Reindex array 113
Remove 103
Remove by index 120
Remove by iterator 100
Remove from export list 51
Remove from scope 50
Remove model 34
Rename model 41
Rename schema instance 32
Repository 20, 29, 31
Reset array index 114
Restore error handler 17, 18
Save changes 45, 46
Schema definition 39, 44
Schema instance 30, 31–40
Scope 48–57
    add 48
    add to export list 51
    copy to other model 53
    copy to same model 53
    delete instance 52
    get export list 56
    owner 49
    remove 50
    remove from export list 51
    validate reference restriction 54
SdaiAccessMode 13, 21
sdaiAccessModel() 42

sdaiAccessModelBN() 42
SdaiADB 9, 14
SdaiADBId 14
sdaiAdd() 101
sdaiAddModel() 33
sdaiAddModelBN() 33
sdaiAddToExportList() 51
sdaiAddToScope() 48
SdaiAggr 10
SdaiAggrIndex 14
SdaiAppInstance 11
SdaiArray 10
SdaiAttr 11
SdaiBag 10
sdaiBeginning() 94
SdaiBinary 9
SdaiBoolean 8
sdaiBreakTrx() 22
sdaiCloseRepository() 31
sdaiCloseSession() 20
SdaiCommitMode 15
sdaiCreateADB() 26
sdaiCreateAggr() 77
sdaiCreateAggrADB() 78
sdaiCreateAggrADBBN() 78
sdaiCreateAggrBN() 77
sdaiCreateComplexInstance() 46
sdaiCreateComplexInstanceBN() 46
sdaiCreateEmptyADB() 26
sdaiCreateInstance() 44
sdaiCreateInstanceBN() 44
sdaiCreateIterator() 93
sdaiCreateModel() 29
sdaiCreateModelBN() 29
sdaiCreateNestedAggr() 102
sdaiCreateNestedAggrADB() 102
sdaiCreateNestedAggrByIndex() 108
sdaiCreateNestedAggrByIndexADB() 108
sdaiCreateNestedAggrByItr() 99
sdaiCreateNestedAggrByItrADB() 99
sdaiCreateNPL() 23
sdaiCreateSchemaInstance() 30
sdaiCreateSchemaInstanceBN() 30
SdaiDefinedType 12
sdaiDeleteADB() 29
sdaiDeleteInstance() 75
sdaiDeleteIterator() 94
sdaiDeleteModel() 40
sdaiDeleteModelBN() 40

sdaiDeleteNPL() 24
sdaiDeleteSchemaInstance() 31
sdaiDeleteSchemaInstanceBN() 32
sdaiEnd() 105
sdaiEndModelAccess() 43
sdaiEndTrx() 22
SdaiEntity 12
SdaiEnum 9
SdaiErrorCode 14
SdaiErrorHandler 14
SdaiErrorId 14
sdaiErrorQuery() 17
SdaiExplicitAttr 11
sdaiFarCopyInstance() 74
sdaiFindInstanceRolesOf() 70
sdaiFindInstanceTypeOf() 71
sdaiFindInstanceUsedIn() 68
sdaiFindInstanceUsedInBN() 68
sdaiFindInstanceUsers() 67
sdaiGetADBType() 28
sdaiGetADBTypePath() 121
sdaiGetADBValue() 26
sdaiGetAggrByIndex() 104
sdaiGetAggrByIterator() 95
sdaiGetAggrElementBoundByIndex() 106
sdaiGetAggrElementBoundByItr() 96
sdaiGetAllAttrs() 73
sdaiGetAttr() 61
sdaiGetAttrBN() 61
sdaiGetAttrBound() 69
sdaiGetAttrBoundBN() 69
sdaiGetAttrDefinition() 61
sdaiGetAttrDefinitionBN() 61
sdaiGetAttrs() 71
sdaiGetAttrsBN() 71
sdaiGetComplexEntity() 57
sdaiGetComplexEntityBN() 57
sdaiGetDescription() 81
sdaiGetEntity() 44
sdaiGetEntityExtent() 47
sdaiGetEntityExtentBN() 47
sdaiGetExportList() 56
sdaiGetInstanceModel() 63
sdaiGetInstanceType() 64
sdaiGetLowerBound() 97
sdaiGetLowerIndex() 110
sdaiGetMemberCount() 91
sdaiGetOwnedScopeInstances() 55
sdaiGetPersistentLabel() 79

sdaiGetSchema() 39
sdaiGetSchemaInstance() 40
sdaiGetScope() 49
sdaiGetScopeOwner() 56
sdaiGetSessionId() 80
sdaiGetUpperBound() 98
sdaiGetUpperIndex() 111
SdaiGlobalRule 12
SdaiId 10, 15
SdaiIndexId 14
sdaiInsertAfter() 115
sdaiInsertBefore() 114
sdaiInsertByIndex() 116
sdaiInsertNestedAggrAfter() 118
sdaiInsertNestedAggrAfterADB() 118
sdaiInsertNestedAggrBefore() 117
sdaiInsertNestedAggrBeforeADB() 117
sdaiInsertNestedAggrByIndex() 119
sdaiInsertNestedAggrByIndexADB() 119
SdaiInstance 10
SdaiInteger 8
sdaiIsDeqWith() 60
sdaiIsDeqWithBN() 60
sdaiIsEqual() 18
sdaiIsInstanceOf() 65
sdaiIsInstanceOfBN() 65
sdaiIsKindOf() 65
sdaiIsKindOfBN() 65
sdaiIsMember() 92
sdaiIsRecordingOn() 25
sdaiIsScopeOwner() 49
sdaiIsSDAIKindOf() 66
sdaiIsSDAIKindOfBN() 66
sdaiIsSDAISubtypeOf() 59
sdaiIsSDAISubtypeOfBN() 59
sdaiIsSubtypeOf() 58
sdaiIsSubtypeOfBN() 58
sdaiIsValidationCurrent() 38
SdaiIterator 13
SdaiItrId 13
SdaiList 10
SdaiLogical 9
sdaiLogicalCompare() 16
SDAI-model 29, 33, 34, 40–48, 63
SdaiModel 11
SdaiNamedType 12
sdaiNearCopyInstance() 73
sdaiNext() 95
SdaiNPL 13

SdaiNullId 15
SdaiNumber 8
sdaiOpenRepository() 20
sdaiOpenRepositoryBN() 21
sdaiOpenSession() 16
SdaiOrderedAggr 10
sdaiPrevious() 105
sdaiPromoteModel() 43
sdaiPutADBTypePath() 121
sdaiPutADBValue() 27
sdaiPutAggrByIndex() 107
sdaiPutAggrByIterator() 100
sdaiPutAllAttrs() 91
sdaiPutAttr() 76
sdaiPutAttrBN() 76
sdaiPutAttrs() 90
sdaiPutAttrsBN() 90
sdaiQuery() 24
SdaiQuerySourceType 13
SdaiReal 8
sdaiRecordEvent() 19
sdaiReindexArray() 113
sdaiRemove() 103
sdaiRemoveByIndex() 120
sdaiRemoveByIterator() 101
sdaiRemoveFromExportList() 51
sdaiRemoveFromScope() 50
sdaiRemoveModel() 34
sdaiRemoveModelBN() 34
sdaiRenameModel() 41
sdaiRenameModelBN() 41
sdaiRenameSchemaInstance() 32
sdaiRenameSchemaInstanceBN() 32
SdaiRep 11
sdaiResetArrayIndex() 114
sdaiRestoreErrorHandler() 18
sdaiSaveChanges() 46
SdaiSchema 12
SdaiSchemaInstance 13
SdaiScope 13
sdaiScopedCopyInSameModel() 53
sdaiScopedCopyToOtherModel() 54
sdaiScopedDelete() 52
SdaiSession 11
SdaiSet 10
sdaiSetErrorHandler() 18
sdaiSetEventRecording() 19
sdaiStartTrx() 21
SdaiString 9

sdaiTestArrayByIndex() 109
sdaiTestArrayByItr() 110
sdaiTestAttr() 62
sdaiTestAttrBN() 62
SdaiTrx 13
sdaiUndoChanges() 45
SdaiUniRule 12
SdaiUnorderedAggr 10
sdaiUnsetADB() 28
sdaiUnsetArrayByIndex() 112
sdaiUnsetArrayByItr() 112
sdaiUnsetAttr() 77
sdaiUnsetAttrBN() 77
sdaiValidateAggrSizes() 85
sdaiValidateAggrUni() 86
sdaiValidateArrayNotOptional() 86
sdaiValidateAttrTypes() 84
sdaiValidateBinaryWidth() 88
sdaiValidateGlobalRule() 35
sdaiValidateGlobalRuleBN() 35
sdaiValidateInverseAttrs() 83
sdaiValidateRealPrecision() 89
sdaiValidateReferenceDomain() 37
sdaiValidateRequiredAttrs() 82
sdaiValidateSchemaInstance() 37
sdaiValidateScopeReferenceRestrictions() 54
sdaiValidateStringWidth() 87
sdaiValidateTypePath() 122
sdaiValidateUniqueness() 36
sdaiValidateUniquenessBN() 36
sdaiValidateWhereRule() 81
sdaiValidateWhereRuleBN() 81
SdaiWhereRule 12
Select type path 120–123
Session 16, 19, 20, 80
Set error handler 17
Set event recording 19
Start transaction 21
String width 69, 87, 96, 106
Test array by index 109
Test array by iterator 109
Test attribute 62, 76
Transaction 15, 21–23
Type path, see select type path
Undo changes 45, 46
Uniqueness 35, 85
Unset ADB 28
Unset array by index 112
Unset array by iterator 112

Unset attribute 76
Upper bound 98
Upper index 111, 113, 114
Validate aggregates size 85
Validate aggregates uniqueness 85
Validate array not optional 86
Validate attribute types 84
Validate binary width 88
Validate global rule 34
Validate inverse attributes 83
Validate real precision 89
Validate reference domain 36
Validate required attributes 82
Validate schema instance 37, 38
Validate scope reference restrictions 54
Validate string width 87
Validate type path 122
Validate uniqueness rule 35
Validate where rule 81
Where rule 81

ICS 25.040.40

Price based on 144 pages