INTERNATIONAL STANDARD

ISO 10303-22

First edition
1998-12-15

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

Part 22:
Implementation methods: Standard data access interface

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

Partie 22: Méthodes de mise en application: Interface normalisée d'accès aux données

Reference number
ISO 10303-22:1998(E)

Contents

page

International Organization for Standardization
Case postale 56 • CH-1211 Genève 20 • Switzerland
Internet iso@iso.ch

Printed in Switzerland

Annexes

Figures page

Tables page

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.

Draft International Standards adopted by 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.

International Standard ISO 10303-22 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data*.

ISO 10303 consists of the following parts under the general title *Industrial automation systems and integration - Product data representation and exchange*:

— Part 1, Overview and fundamental principles;

— Part 11, Description method: EXPRESS language reference manual;

— Part 12, Description method: EXPRESS-I language reference manual;

— Part 21, Implementation method: Clear text encoding of the exchange structure;

— Part 22, Implementation method: Standard data access interface specification;

— Part 23, Implementation method: C++ language binding to the standard data access interface;

— Part 24, Implementation method: C language binding to the standard data access interface;

— Part 26, Implementation method: Interface definition language binding to the standard data access;

— Part 31, Conformance testing methodology and framework: General concepts;

— Part 32, Conformance testing methodology and framework: Requirements on testing laboratories and clients;

— Part 33, Conformance testing methodology and framework: Structure and use of abstract test suites;

— Part 34, Conformance testing methodology and framework: Abstract test methods;

— Part 35, Conformance testing methodology and framework: Abstract test methods for SDAI implementations;

— Part 41, Integrated generic resource: Fundamentals of product description and support;

— Part 42, Integrated generic resource: Geometric and topological representation;

— Part 43, Integrated generic resource: Representation structures;

— Part 44, Integrated generic resource: Product structure configuration;

— Part 45, Integrated generic resource: Materials;

— Part 46, Integrated generic resource: Visual presentation;

— Part 47, Integrated generic resource: Shape variation tolerances;

— Part 49, Integrated generic resource: Process structure and properties;

— Part 101, Integrated application resource: Draughting;

— Part 104, Integrated application resource: Finite element analysis;

— Part 105, Integrated application resource: Kinematics;

— Part 106, Integrated application resource: Building construction core model;

— Part 201, Application protocol: Explicit draughting;

— Part 202, Application protocol: Associative draughting;

— Part 203, Application protocol: Configuration controlled design;

— Part 204, Application protocol: Mechanical design using boundary representation;

— Part 205, Application protocol: Mechanical design using surface representation;

— Part 207, Application protocol: Sheet metal die planning and design;

— Part 208, Application protocol: Life cycle management - Change process;

— Part 209, Application protocol: Composite and metallic structural analysis and related design;

— Part 210, Application protocol: Electronic assembly, interconnect, and packaging design;

— Part 212, Application protocol: Electrotechnical design and installation;

— Part 213, Application protocol: Numerical control process plans for machined parts;

— Part 214, Application protocol: Core data for automotive design;

— Part 215, Application protocol: Ship arrangement;

— Part 216, Application protocol: Ship moulded forms;

— Part 217, Application protocol: Ship piping;

— Part 218, Application protocol: Ship structures;

— Part 220, Application protocol: Process planning, manufacture, and assembly of layered electronic products;

— Part 221, Application protocol: Functional data and their schematic representation for process plant;

— Part 222, Application protocol: Exchange of product data for composite structures;

— Part 223, Application protocol: Exchange of design and manufacturing product information for cast parts;

— Part 224, Application protocol: Mechanical product definition for process plans using machining features;

— Part 225, Application protocol: Building elements using explicit shape representation;

— Part 226, Application protocol: Ship mechanical systems;

— Part 227, Application protocol: Plant spatial configuration;

— Part 228, Application protocol: Building services: Heating, ventilation, and air conditioning;

— Part 229, Application protocol: Exchange of design and manufacturing product information for forged parts;

— Part 230, Application protocol: Building structural frame: Steelwork;

— Part 231, Application protocol: Process engineering data: Process design and process specification of major equipment;

— Part 232, Application protocol: Technical data packaging core information and exchange;

— Part 301, Abstract test suite: Explicit draughting;

— Part 302, Abstract test suite: Associative draughting;

— Part 303, Abstract test suite: Configuration controlled design;

— Part 304, Abstract test suite: Mechanical design using boundary representation;

— Part 305, Abstract test suite: Mechanical design using surface representation;

— Part 307, Abstract test suite: Sheet metal die planning and design;

— Part 308, Abstract test suite: Life cycle management - Change process;

— Part 309, Abstract test suite: Composite and metallic structural analysis and related design;

— Part 310, Abstract test suite: Electronic assembly, interconnect, and packaging design;

— Part 312, Abstract test suite: Electrotechnical design and installation;

— Part 313, Abstract test suite: Numerical control process plans for machined parts;

— Part 314, Abstract test suite: Core data for automotive mechanical design processes;

— Part 315, Abstract test suite: Ship arrangement;

— Part 316, Abstract test suite: Ship moulded forms;

— Part 317, Abstract test suite: Ship piping;

— Part 318, Abstract test suite: Ship structures;

— Part 320, Abstract test suite: Process planning, manufacture, and assembly of layered electronic products;

— Part 321, Abstract test suite: Functional data and their schematic representation for process plant;

— Part 322, Abstract test suite: Exchange of product data for composite structures;

— Part 323, Abstract test suite: Exchange of design and manufacturing product information for cast parts;

— Part 324, Abstract test suite: Mechanical product definition for process plans using machining features;

— Part 325, Abstract test suite: Building elements using explicit shape representation;

— Part 326, Abstract test suite: Ship mechanical systems;

— Part 327, Abstract test suite: Plant spatial configuration;

— Part 328, Abstract test suite: Building services: Heating, ventilation, and air conditioning;

— Part 329, Abstract test suite: Exchange of design and manufacturing product information for forged parts;

— Part 330, Abstract test suite: Building structural frame: Steelwork;

— Part 331, Abstract test suite: Process engineering data: Process design and process specification of major equipment;

— Part 332, Abstract test suite: Technical data packaging core information and exchange;

— Part 501, Application interpreted construct: Edge-based wireframe;

— Part 502, Application interpreted construct: Shell-based wireframe;

— Part 503, Application interpreted construct: Geometrically bounded 2D wireframe;

— Part 504, Application interpreted construct: Draughting annotation;

— Part 505, Application interpreted construct: Drawing structure and administration;

— Part 506, Application interpreted construct: Draughting elements;

— Part 507, Application interpreted construct: Geometrically bounded surface;

— Part 508, Application interpreted construct: Non-manifold surface;

— Part 509, Application interpreted construct: Manifold surface;

— Part 510, Application interpreted construct: Geometrically bounded wireframe;

— Part 511, Application interpreted construct: Topologically bounded surface;

— Part 512, Application interpreted construct: Faceted boundary representation;

— Part 513, Application interpreted construct: Elementary boundary representation;

— Part 514, Application interpreted construct: Advanced boundary representation;

— Part 515, Application interpreted construct: Constructive solid geometry;

— Part 517, Application interpreted construct: Mechanical design geometric presentation;

— Part 518, Application interpreted construct: Mechanical design shaded representation.

The structure of this International Standard is described in ISO 10303-1. The numbering of the parts of the International Standard reflects its structure:

— Parts 11 to 13 specify the description methods,

— Parts 21 to 26 specify the implementation methods,

— Parts 31 to 35 specify the conformance testing methodology and framework,

— Parts 41 to 49 specify the integrated generic resources,

— Parts 101 to 106 specify the integrated application resources,

— Parts 201 to 232 specify the application protocols,

— Parts 301 to 332 specify the abstract test suites, and

— Parts 501 to 518 specify the application interpreted constructs.

Should further parts be published, they will follow the same numbering pattern.

Annexes A, B and C form an integral part of this part of ISO 10303. Annexes D and E are for information only.

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

This International Standard is organized as a series of parts, each published separately. The parts of ISO 10303 fall into one of the following series: description methods, integrated resources, application interpreted constructs, application protocols, abstract test suites, implementation methods, and conformance testing. The series is described in ISO 10303-1. This part of ISO 10303 is a member of the implementation methods series.

This part of ISO 10303 specifies the standard data access interface (SDAI) to data defined using ISO 10303-11 (EXPRESS). Operations are defined that give the application programmer the capability to manipulate data through this interface based upon its description in the defining schema or schemas. The standardization of a data access interface along with data definitions facilitates integration of different software components from different vendors.

Major subdivisions in this part of ISO 10303 are:

— SDAI environment constructs defined using EXPRESS found in clauses 6 through 9;

— SDAI operations, errors and state found in clauses 10 through 12;

— Implementation classes of the SDAI functionality to which implementations claim conformance found in clause 13.

Computer application systems are implemented using computing languages. The specification of the functionality defined in this part of ISO 10303 in a particular computing language is referred to as an SDAI language binding. Since there are many computing languages, many SDAI language bindings are possible. SDAI language bindings are specified as other parts of ISO 10303 within the implementation method series.

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

International automation systems and integration – Product data representation and exchange – Part 22: Implementation methods: Standard data access interface

1 Scope

This part of ISO 10303 specifies the functional characteristics of a data access interface. This interface is referred to as the standard data access interface (SDAI). The SDAI specifies the operations available to an application for the purposes of acquiring and manipulating data whose structure is defined using ISO 10303-11 (EXPRESS).

The SDAI is specified in terms independent of any computing language or system. The specification of the functionality defined by the SDAI in a particular computing language is referred to as an SDAI language binding. SDAI language bindings are specified as companion documents within the implementation methods series of ISO 10303.

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

— access to and manipulation of instances of entities described using the EXPRESS data specification language;

— access to multiple data repositories by a single application at the same time;

— capabilities for an application to organize operations into groups whose effect can be saved or cancelled at the discretion of the application;

— access to a dictionary describing the data elements that can be manipulated by an application;

— ability to invoke the validation of the constraints specified using EXPRESS at the discretion of the application;

— support for managing the dependency relationships between entity instances;

— capabilities to describe logical collections of entity instances that define the population over which entity instance to entity instance references are allowed;

— capabilities to describe logical collections of entity instances that define the population over which global rules are validated;

— support for the use of data created within the context of one schema in the context of another schema.

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

— the complete specification of the behaviour of an SDAI implementations in a multi-user environment;

NOTE 1 - An SDAI implementation is not precluded from providing multi-user data sharing access where the behaviour of the implementation depends on the underlying data storage technology.

— specific support for establishing a connection to a remote data repository;

NOTE 2 - An SDAI implementation is not precluded from providing access to a remote data repository via some other mechanism.

— data access and manipulation operations that are specific to the semantics of the data;

— specification of the mechanisms or formats by which data is held in a data repository;

— creation, deletion, and naming of the data repositories available via the SDAI.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO 10303. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO 10303 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of the IEC and ISO maintain registers of currently valid International Standards.

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 10303-31:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 31: Conformance testing methodology and framework: General concepts.*

ISO/IEC 8824-1:1995, *Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

ISO 8601:1988, *Data elements and interchange formats - Information interchange - Representation of dates and times.*

3 Definitions and abbreviations

For the purposes of this part of ISO 10303, the following definitions and abbreviations apply.

3.1 Terms defined in ISO 10303-1

This part of ISO 10303 makes use of the following terms defined in ISO 10303-1:

— application;

— application protocol (AP);

— data;

— implementation method;

— information;

— information model;

— product information model;

— protocol implementation conformance statement (PICS);

— structure.

3.2 Terms defined in ISO 10303-11

This part of ISO 10303 makes use of the following terms defined in ISO 10303-11:

— complex entity data type;

— data type;

— entity;

— entity data type;

— entity instance;

— instance;

— population;

— value.

3.3 Other definitions

For the purpose of this part of ISO 10303, the following definitions apply.

3.3.1 application schema: an information model, defined using EXPRESS, describing the data of interest in a particular context or domain.

NOTE - The schemas defined in the ISO 10303 application protocol series of parts would be considered application schemas.

3.3.2 concurrent access: the ability for more than one computer application to manipulate data in a repository at the same time.

3.3.3 constraint: a limitation, defined using EXPRESS, imposed on data against which the data may be evaluated to determine whether the data is valid within a particular context.

3.3.4 current schema: an EXPRESS schema into which items from other schemas are made visible via the EXPRESS interface specification.

3.3.5 external schema: an EXPRESS schema, with all interfaced items resolved, containing data types that are defined as domain equivalent with data types in a native schema.

NOTE - A.1.1 describes the process for resolving interfaced items.

3.3.6 foreign schema: an EXPRESS schema other than the current schema from which items are made visible in the current schema via the EXPRESS interface specification.

3.3.7 identifier: an implementation-dependent name by which entity instances and aggregate instances are uniquely known within an SDAI session.

3.3.8 implementation class: the specification of an implementable subset of the functionality defined in this part of ISO 10303 to which an implementation may conform.

3.3.9 iterator: a mechanism that allows an application program to traverse the contents of an instance of an aggregate.

3.3.10 native schema: an EXPRESS schema, with all interfaced items resolved, upon which a schema instance or SDAI-model may be based.

3.3.11 repository: an identifiable data storage facility.

3.3.12 schema instance: a logical association of related SDAI-models establishing a domain of entity instances. This domain is the limit for references between entity instances and is the domain over which global rule validation occurs.

3.3.13 SDAI-model: a container within which related entity instances exist.

3.3.14 SDAI language binding: the SDAI functionality specified in a particular computing language.

3.3.15 SDAI schema: an EXPRESS schema that is defined within this part of ISO 10303.

3.3.16 session: the operations that occur between the initiation and termination of the use of an SDAI implementation by one application.

3.3.17 validation: the process of checking that instances satisfy the constraints specified in the schema that describe their structure, values, and relationships.

3.4 Abbreviations

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

AP	application protocol
PICS	protocol implementation conformance statement
RO	read-only
RW	read-write
SDAI	standard data access interface

4 SDAI overview

4.1 Data access interfaces

EXPRESS allows the specification of entities with attributes and the constraints that a valid population of those entities meets. The SDAI specifies the requirements of a programming interface for the creation and manipulation of instances of EXPRESS entities. The SDAI and EXPRESS together specify a data access interface that is independent of the underlying storage technology.

4.2 Operations and the session state

Once an SDAI session has been initiated, SDAI operations may be used to manipulate instances of entity types defined in application and SDAI schemas. The session has several states expressed in a tabular form in clause 12. In each state a set of SDAI operations, some of which may have the effect of changing the state, are available. Information concerning the session and its state is available as a population of the SDAI session schema and SDAI population schema (see clauses 7 and 8) during the session.

4.3 Repositories, schema instances, and SDAI-models

The SDAI defines an interface between an application and the environment in which entity instances exist. Two aspects of that environment are known as repositories and as schema instances. Repositories are data storage facilities. Schema instances are logical collections of SDAI-models from which a set of entity instances can be derived. This set of entity instances is the domain over which references between entity instances and global rule validation are supported. While schema instances, like SDAI-models, are created

within a repository, SDAI-models from any repository may be associated with the schema instance.

NOTE - A repository may be implemented in memory, as a single database, multiple databases, a single file, a collection of files, or any other form.

EXAMPLE 1 - Figure 1 depicts several of the relationships between SDAI-models, repositories and schema instances. Schema instances 1A and 1B and SDAI-models 11, 12 and 13 are based upon Schema 1 yet they exist in separate repositories. References between SDAI-model 11 and 13 are not allowed as the two SDAI-models are not associated with the same schema instance. SDAI-model 13 based upon Schema 1 is associated with Schema instance 2A based upon Schema 2. For this to be allowed, at least one pair of entity types was declared as domain equivalent between the two schemas in the SDAI data dictionary.

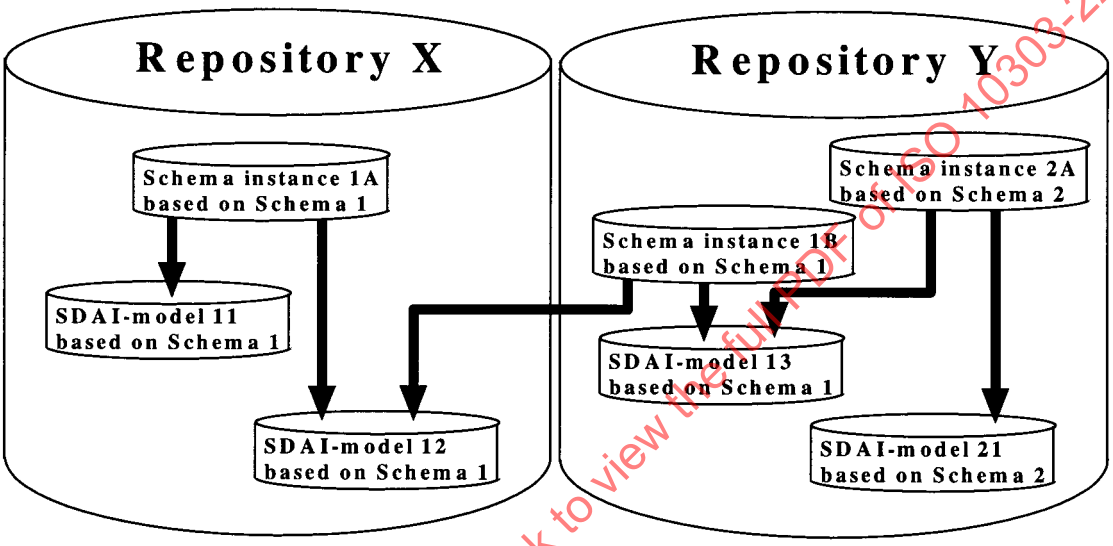

Figure 1 - An example SDAI storage structure

Entity instances are created within SDAI-models that are created within repositories. The entity instances making up each SDAI-model are based on a single EXPRESS schema with the interface specifications resolved. Entity instances within one SDAI-model may refer to entity instances within another SDAI-model, provided that there exists a schema instance with which both SDAI-models are associated. The two SDAI-models shall either be based upon the same EXPRESS schema or be based upon two EXPRESS schemas that were defined as having domain equivalent constructs (see A.2). An SDAI-model may be associated with more than one schema instance.

4.4 Transactions and access modes

Implementation levels of the SDAI are defined that support transactions. A transaction consists of a series of operations whose effect may be saved or undone as a unit. Facilities are also available enabling the application to manage access to specific repositories and SDAI-models. Transactions and

access to SDAI-models have modes associated with them: read-write and read-only. The read-write mode enables operations that access, create, update and delete the instances in the SDAI-models and repositories. The read-only mode disables operations that create, update or delete instances in the SDAI-models and repositories. An SDAI-model cannot be made available in read-write mode when a transaction initiated as read-only mode is in effect.

4.5 The session, data dictionary and managing a population

The SDAI session schema (see clause 7) describes the structure of an SDAI session. The SDAI population schema (see clause 8) describes the organizing structures available for managing a population based upon a schema. The SDAI population schema specifies the organizational objects that an application may create during the course of a session.

In order to support applications that require access to the information about the schema defining the application data, the SDAI specifies a data dictionary. The SDAI dictionary schema (see clause 6) describes the structure of the data dictionary. The collection of instances of the entities specified in the SDAI dictionary schema constitutes the data dictionary. As not all applications require access to a data dictionary, an implementation class is specified with no support of the data dictionary required (see 13.1.2). In this case the application programmer needs complete knowledge of the schema and makes reference to items from the schema by name.

NOTE - Figure 2 shows the relationships between application data, dictionary, session and population organizing data, the application program and the SDAI implementation in simplified terms. Figures 2 also shows to which types of data the application program has read-write or read-only access.

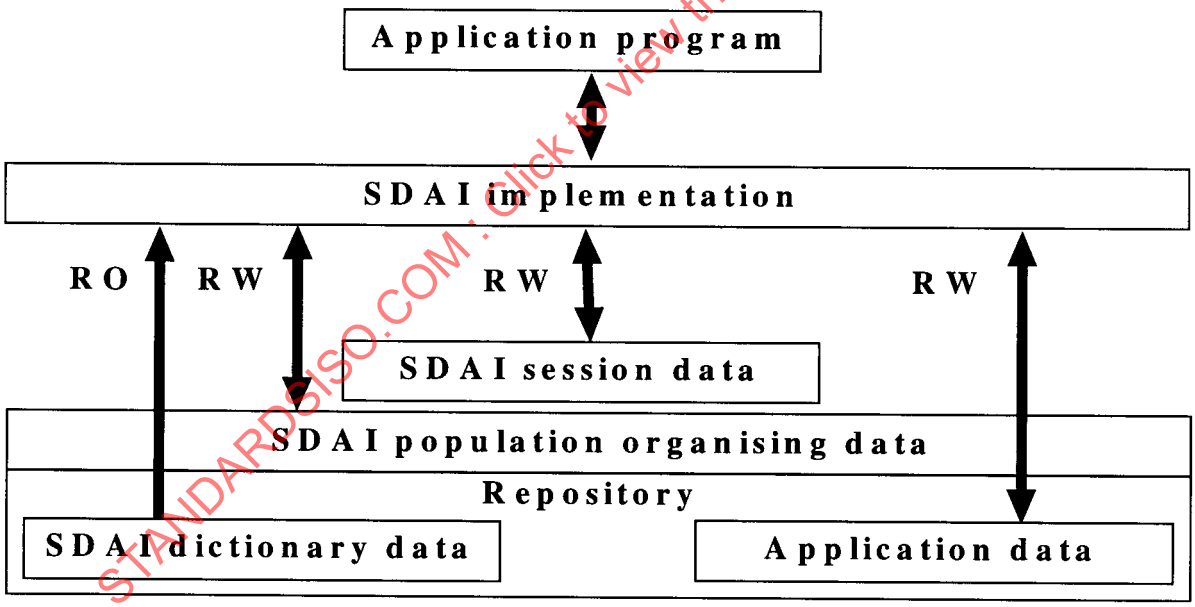

Figure 2 - The SDAI data architecture element relationships

The creation, deletion and modification of the session, population organizing and dictionary data occurs as the consequence of special operations whose purpose is to manage the SDAI environment. The entity instance operations are used to access the session, population organizing and dictionary data in the same manner as application instances. Not all entity instance operations are required to support access to the population organizing data as the instances of the SDAI population schema are not required to exist within an SDAI-model (see 8.1).

4.6 SDAI parameter data schema

The SDAI parameter data schema (see clause 9) describes, in abstract terms, the various types of instance data that are passed through the interface. It is provided to enable the specification of the SDAI operations. The complete SDAI parameter data schema need not be implemented by an SDAI implementation. The SDAI parameter data schema does define the subtype relationships between the types of entity instances that an SDAI implementation shall support. Other characteristics of these instances are not specified in this schema as they are implementation-specific.

4.7 Functional specification

The SDAI defines the functional specification for a set of operations for acquiring and manipulating data. The SDAI operations are divided into several categories:

— environment operations that establish an SDAI session (see 10.3);

— session operations that allow an application to manage the transactions, repositories, and queries in the session (see 10.4);

— repository operations that allow an application to manage access to SDAI-models within the repositories (see 10.5);

— schema instance operations that allow an application to manage the association of SDAI-models with schema instances and validate EXPRESS global rules and references within a schema instance (see 10.6);

— SDAI-model operations that allow an application to create instances and manage the SDAI-model access (see 10.7);

— operations that allow an application to create and manage support for the dependency relationships between entity instances (see 10.8);

NOTE - These dependency relationships are based upon the SCOPE construct described in ISO 10303-21.

— operations that allow an application to examine type information and domain equivalency information (see 10.9);

— entity instance operations that allow an application to manipulate instances of entity data types found in both SDAI-defined schemas and application schemas (see 10.10);

— application instance operations that allow an application to create, modify, delete and validate instances of entity data types and create instances of aggregates defined in application schemas (see 10.11);

— categories of entity and application instance operations that allow an application to create, delete, manipulate and modify instances of various aggregate types (see 10.12 through 10.19).

The operations are categorized such that all operations manipulating a particular kind of object are described in the subclause dealing with that object. The exceptions to this categorization are the operations that create, open or start the various SDAI and application objects. As these operations usually affect properties of higher level objects, they are considered operations upon the object within which the current object is being created, opened or started.

EXAMPLE 2 - Opening a repository is a session operation, not a repository operation, since it modifies properties of the session.

The specification of each SDAI operation lists the required input and output parameters. The types of these parameters are drawn from the SDAI dictionary, session, population and the parameter data schemas.

4.8 SDAI language bindings

This part of ISO 10303 specifies operations independently of any programming language. SDAI language bindings of these operations are developed for computing languages to define the capabilities required of implementations. SDAI language bindings are published as other parts of the implementation methods series of ISO 10303. An SDAI implementation provides the operations described in a particular SDAI language binding.

SDAI language bindings support the operations described in clause 10 but there need not be a one-to-one correspondence between the operations described there and the functions or subroutines defined in the SDAI language binding. An SDAI language binding may extend or add to the functionality defined in clause 10 to provide for a more efficient or convenient implementation of those operations.

Two types of SDAI language bindings are identified: late bindings and early bindings.

Late bindings are applied to any application schema in the same manner using one set of functions. This set is available to the application programmer independently of the application schema. References to particular EXPRESS constructs are provided via parameters passed as input to the SDAI operations. A late binding specifies the set of SDAI operations to be made available to the application programmer explicitly.

NOTE 1 - These operations and their parameters are typically evaluated at run-time to execute the desired function.

EXAMPLE 3 - An operation to set the value of attribute "a1" of entity instance "i1" of entity type "t1" might be written in a late binding as SetValue(i1, t1.a1, value).

Early bindings describe the production of a data access interface based upon a specific application schema. Instead of being provided via parameters passed as input to the SDAI operations, references to

particular EXPRESS constructs may be implicitly or explicitly embedded in the function or subroutine names. The complete set of functions available to the application programmer depends upon the application schema underlying the SDAI implementation.

NOTE 2 - The application schema may be compiled to produce the implementation. These compiler produced functions and their parameters are typically evaluated at application program compile-time.

EXAMPLE 4 - An operation to set the value of attribute "a1" of entity instance "i1" of entity type "t1" might be written in an early binding as SetValuet1a1(i1, value).

Early and late SDAI language bindings may be published together as one part in the implementation methods series of ISO 10303. SDAI implementations may provide support for both early and late SDAI language bindings and may support the use of both within a single application program.

4.9 Error handling

The description of each operation includes a list of possible error indicators for conditions which may arise, preventing the operations successful completion. This part of ISO 10303 does not specify how errors shall be reported by individual SDAI operations. Each SDAI language binding defines an error reporting mechanism appropriate to its target language.

The SDAI session schema defines an error event record, via the **sdai_session.errors** attribute, that is available as a resource to any SDAI language binding error handling mechanism and to the application programmer.

SDAI language bindings map the complete list of error indicators in clause 11 to an appropriate construct in the target language. An implementation shall set the error codes from clause 11 in the **error_event.error** attribute (see 7.4.7) when an SDAI operation fails except for the Open Session operation (see 10.3.1) which cannot set the attribute as a session does not yet exist. The error reporting mechanism defined in each SDAI language binding need not return error codes identical in type or value to the error codes set in the **error_event.error** attribute. A binding may specify additional functions or subroutines for error reporting and handling. Any such function or subroutine does not change the state of the SDAI session errors.

5 Fundamental principles

The following concepts and assumptions apply:

— During the course of an SDAI session, a unique and unchanging identifier is available for each instance of an entity or aggregation described in an SDAI or application schema;

— The EXPRESS schema upon which SDAI-models and schema instances are based is the completely expanded form of a schema in that all items interfaced from other schemas are resolved into the referencing schema making the schema complete with no references to other schemas (see A.1);

— The process that makes the population of the SDAI dictionary schema available in a session expands any complex entity data types that result from either an explicit or implicit ANDOR or AND constraint following the approach given in A.1.3;

— When required by the implementation class, SDAI implementations make available population of the SDAI dictionary schema with the EXPRESS items specified in the SDAI session schema and application schemas;

— The schemas defined in clauses 6 through 9 are not application protocols or product information models and as such take efficiency, convenience and the SDAI operations into account;

— Entity instances based upon one schema may be shared in the context of other schemas based upon domain equivalence (see A.2) and the domain equivalence constructs defined in the SDAI dictionary schema (see 6.4.1, 6.4.8 and 6.4.9);

— EXPRESS constraints are validated only at the request of the application. When validation occurs, all creation, modification and deletion of SDAI-models, schema instances, SCOPE constructs and application instances is taken into account.

6 SDAI dictionary schema

The following EXPRESS declaration begins the SDAI dictionary schema.

EXPRESS specification:

```
*)
SCHEMA SDAI_dictionary_schema;
(*
```

6.1 Introduction

The SDAI dictionary schema defines the structure of a data dictionary that makes available information about the EXPRESS schemas that describe the instances operated on in an SDAI session. The structure of the SDAI dictionary schema reflects the structure of EXPRESS itself. Not all items that may be defined using EXPRESS appear in the SDAI dictionary schema as some kinds of items are not required to define the SDAI operations. Mapping EXPRESS schemas into a population of the SDAI dictionary schema is described in annex A.

The population of the SDAI dictionary schema based upon a schema known to the SDAI implementation as part of the data dictionary shall exist in its own separate SDAI-model. Application schemas and the SDAI session schema shall appear as part of the data dictionary.

There shall be a schema instance based upon the SDAI session schema with which the SDAI-models constituting the data dictionary for both application schemas and the SDAI session schema shall be associated. The SDAI-models constituting the SDAI data dictionary shall be assigned read-only access.

An SDAI implementation need not store the dictionary data as described. The SDAI operations accessing the data dictionary shall, however, respond as if the data is structured as described.

6.2 Fundamental concepts and assumptions

The structure of the entities and attributes of the SDAI dictionary schema take implementation and access efficiency into account.

EXAMPLE 5 - The **schema_definition.entities** collection includes all entity data types defined locally in the current schema as well as those interfaced into the current schema.

The SDAI dictionary schema supports the retention of the information that is contained in the EXPRESS interface specification (see A.1.1).

As the SDAI dictionary schema is designed to be populated based upon valid EXPRESS schemas as defined in ISO 10303-11, all the constraints governing the population of the SDAI dictionary schema are not specified in the schema. The constraints specified in ISO 10303-11 shall be enforced on a population of the SDAI dictionary schema. In particular, the constraints on EXPRESS identifiers for items declared in a schema, including the schema itself, are not specified in the SDAI dictionary schema (see 6.3.6).

6.3 SDAI dictionary schema type definitions

This subclause contains the dictionary concepts defined using the EXPRESS TYPE construct.

6.3.1 base_type

A **base_type** is a selection of a **simple_type**, an **aggregation_type**, or a **named_type**. These are the data types that may be used as the value for an attribute or as an element of an aggregate.

EXPRESS specification:

```
*)
TYPE base_type = SELECT
  (simple_type,
  aggregation_type,
  named_type);
END_TYPE;
(*
```

6.3.2 constructed_type

A **constructed_type** is a selection of an **enumeration_type** or a **select_type**. These are data types with a syntactic structure that are used to provide a representation of an EXPRESS defined data type.

EXPRESS specification:

```
*)
TYPE constructed_type = SELECT
  (enumeration_type,
  select_type);
END_TYPE;
```

```
(*
```

6.3.3 underlying_type

An **underlying_type** is a selection of a **simple_type**, an **aggregation_type**, a **defined_type**, or a **constructed_type**. These are the data types that are used to provide the representation of an EXPRESS defined data type.

EXPRESS specification:

```
*)
TYPE underlying_type = SELECT
  (simple_type,
  aggregation_type,
  defined_type,
  constructed_type);
END_TYPE;
(*
```

6.3.4 type_or_rule

A **type_or_rule** is a selection of a **named_type** or a **global_rule**. These declarations establish or constrain a population.

EXPRESS specification:

```
*)
TYPE type_or_rule =  SELECT
  (named_type,
  global_rule);
END_TYPE;
(*
```

6.3.5 explicit_or_derived

An **explicit_or_derived** is a selection of an **explicit_attribute** or a **derived_attribute**. These are the attributes that may be redeclared by a derived attribute (see ISO 10303-11: 9.2.3.4).

EXPRESS specification:

```
*)
TYPE explicit_or_derived = SELECT
  (explicit_attribute,
  derived_attribute);
END_TYPE;
(*
```

6.3.6 express_id

An **express_id** is an EXPRESS identifier (see ISO 10303-11: 7.4) for an item declared in an EXPRESS schema. Although ISO 10303-11 states that the case of letters is not significant for EXPRESS identifiers, lower case letters shall be used as the values of the attributes of the SDAI dictionary schema whose domain is defined by an **express_id**.

EXPRESS specification:

```
*)
TYPE express_id = STRING;
END_TYPE;
(*
```

6.3.7 info_object_id

An **info_object_id** is an unambiguous information object identifier for an EXPRESS schema in an open system (see annex C).

EXPRESS specification:

```
*)
TYPE info_object_id = STRING;
END_TYPE;
(*
```

6.4 SDAI dictionary schema entity definitions

This subclause contains the dictionary concepts defined using the EXPRESS ENTITY construct.

6.4.1 schema_definition

A **schema_definition** is the representation of an EXPRESS SCHEMA and is the construct upon which SDAI-models and schema instances are based. It defines the scope for a collection of entity, type and rule definitions consisting of those definitions found in the current EXPRESS schema and those definitions that are resolved into the current schema as the result of the EXPRESS interface specification. Items from foreign schemas are resolved into the current schema as described in A.1.1.

EXPRESS specification:

```
*)
ENTITY schema_definition;
  name           : express_id;
  identification : OPTIONAL info_object_id;
INVERSE
  entities         : SET [0:?] OF entity_definition FOR parent_schema;
  types            : SET [0:?] OF defined_type FOR parent_schema;
  global_rules     : SET [0:?] OF global_rule FOR parent_schema;
  external_schemas : SET [0:?] OF external_schema FOR native_schema;
UNIQUE
  UR1 : identification;
END_ENTITY;
(*
```

Attribute definitions:

name: the name of the schema.

identification: if present, the information object identifier of the schema upon which the **schema_definition** is based.

entities: the entities declared in or resolved into the schema.

types: the types declared in or resolved into the schema.

global_rules: the global rules declared in or resolved into the schema.

external_schemas: the schemas containing types defined as domain equivalent with types in the schema (see A.2).

Formal propositions:

UR1: the object identification for a schema shall be unique.

6.4.2 interface_specification

An **interface_specification** is the representation of items in the current schema originally declared in a foreign schema (see ISO 10303-11 clause 11). All items interfaced from a particular foreign schema, whether explicitly via USE and/or REFERENCE or implicitly, shall appear in the same entity instance of **interface_specification**. A USE or REFERENCE to an entire schema explicitly interfaces all items declared in that schema.

EXPRESS specification:

```
*)
ENTITY interface_specification;
  current_schema_id : express_id;
  explicit_items    : SET [1:?] OF explicit_item_id;
  implicit_items    : SET [0:?] OF implicit_item_id;
END_ENTITY;
(*
```

Attribute definitions:

current_schema_id: the name of the current schema into which items are interfaced; the schema where the USE or REFERENCE specification is declared.

explicit_items: the items interfaced via EXPRESS USE or REFERENCE specification.

implicit_items: the items implicitly interfaced.

6.4.3 interfaced_item

An **interfaced_item** is an item defined in a foreign schema made available to the current schema via the EXPRESS interface specification.

EXPRESS specification:

```
*)
ENTITY interfaced_item
  ABSTRACT SUPERTYPE OF (ONEOF (explicit_item_id, implicit_item_id));
```

```
  foreign_schema_id : express_id;
END_ENTITY;
(*
```

Attribute definitions:

foreign_schema_id: the name of the schema from which the item is interfaced.

6.4.4 explicit_item_id

An **explicit_item_id** is a **named_type** defined in a foreign schema made available to the current schema explicitly via the EXPRESS USE or REFERENCE construct.

EXPRESS specification:

```
*)
ENTITY explicit_item_id
  ABSTRACT SUPERTYPE OF (ONEOF (used_item, referenced_item))
  SUBTYPE OF ( interfaced_item );
  local_definition : named_type;
  original_id      : OPTIONAL express_id;
END_ENTITY;
(*
```

Attribute definitions:

local_definition: the definition in the current schema of the named type that was interfaced from the foreign schema.

original_id: if present, the name of the type in the foreign schema. The type was renamed in the interface specification.

6.4.5 used_item

A **used_item** is an **explicit_item_id** made available to the current schema via the EXPRESS USE interface specification.

EXPRESS specification:

```
*)
ENTITY used_item
  SUBTYPE OF ( explicit_item_id );
END_ENTITY;
(*
```

6.4.6 referenced_item

A **referenced_item** is an **explicit_item_id** made available to the current schema via the EXPRESS REFERENCE interface specification.

EXPRESS specification:

```
*)
ENTITY referenced_item
  SUBTYPE OF ( explicit_item_id );
END_ENTITY;
(*
```

6.4.7 implicit_item_id

An **implicit_item_id** is a **named_type** or **global_rule** implicitly interfaced into the current schema as the result of an EXPRESS USE or REFERENCE interface specification (see ISO 10303-11: 11.4).

EXPRESS specification:

```
*)
ENTITY implicit_item_id
  SUBTYPE OF ( interfaced_item );
  local_definition : type_or_rule;
END_ENTITY;
(*
```

Attribute definitions:

local_definition: the definition in the current schema of the EXPRESS item that was implicitly interfaced from the foreign schema.

6.4.8 external_schema

An **external_schema** is an EXPRESS schema from which types are declared as domain equivalent with types from the native schema (see A.2).

EXPRESS specification:

```
*)
ENTITY external_schema;
  name          : express_id;
  native_schema : schema_definition;
INVERSE
  for_types : SET [1:?] OF domain_equivalent_type FOR owner;
END_ENTITY;
(*
```

Attribute definitions:

name: the name of the external schema.

native_schema: the local schema with which the types defined in the external schema are domain equivalent.

for_types: the types in the native and external schemas being defined as domain equivalent.

Informal propositions:

IP1: For each schema that contains entity types declared domain equivalent with entity types in a native schema, a separate instance of external schema shall exist.

6.4.9 domain_equivalent_type

A **domain_equivalent_type** associates the name of a **named_type** from an external schema with a **named_type** in the native schema. This declares that the **named_type** defined in the external schema is domain equivalent with the **named_type** defined in the native schema.

EXPRESS specification:

```
*)
ENTITY domain_equivalent_type;
  external_type_id : express_id;
  native_type      : named_type;
  owner            : external_schema;
END_ENTITY;
(*
```

Attribute definitions:

external_type_id: the name of the type in an external schema for which domain equivalence with the type in the native schema is being declared.

native_type: the definition of the type in the native schema for which a domain equivalent type is being declared.

owner: the external schema owning the definition of the current domain equivalent type.

Informal propositions:

IP1: An instance of **domain_equivalent_type** shall exist for each type in the external schema that is to be domain equivalent with a native type.

IP2: In the case where the native type or external type or both is a **defined_type**, the **underlying_type** of the **defined_type** shall evaluate to be a **select_type** with at least one entity type included as a select item.

6.4.10 named_type

A **named_type** is an EXPRESS data type that has a name and that may have applicable domain rules.

EXPRESS specification:

```
*)
ENTITY named_type
  ABSTRACT SUPERTYPE OF (ONEOF(entity_definition, defined_type));
  name          : express_id;
  where_rules   : LIST [0:?] OF where_rule;
  parent_schema : schema_definition;
```

```
END_ENTITY;
(*
```

Attribute definitions:

name: the name of the data type.

where_rules: the domain rules defined in the data type declaration in the order in which they appear in that declaration.

parent_schema: the **schema_definition** with which the **named_type** is associated in the data dictionary.

6.4.11 defined_type

A **defined_type** is a **named_type** establishing a type as the result of the EXPRESS TYPE declaration. A **defined_type** has a name and a domain.

EXPRESS specification:

```
*)
ENTITY defined_type
  SUBTYPE OF (named_type);
  domain : underlying_type;
END_ENTITY;
(*
```

Attribute definitions:

domain: the underlying type of the defined type.

6.4.12 entity_definition

An **entity_definition** is a **named_type** establishing an entity as the result of an EXPRESS ENTITY declaration or as the result of the mapping applied to a combination of EXPRESS ENTITY declarations constrained using the EXPRESS ANDOR or AND keyword (see A.1.3). Entities established by this mapping are considered subtypes of the constituent entity types.

EXPRESS specification:

```
*)
ENTITY entity_definition
  SUBTYPE OF (named_type);
  supertypes    : LIST [0:?] OF UNIQUE entity_definition;
  complex       : BOOLEAN;
  instantiable  : BOOLEAN;
  independent   : BOOLEAN;
INVERSE
  attributes        : SET [0:?] OF attribute FOR parent_entity;
  uniqueness_rules  : SET [0:?] OF uniqueness_rule FOR parent_entity;
  global_rules      : SET [0:?] OF global_rule FOR entities;
END_ENTITY;
(*
```

Attribute definitions:

supertypes: the list of entity types of which the entity type is an immediate SUBTYPE, ordered alphabetically based upon the value of the **entity_definition.name**. If the entity type is the result of the mapping of the EXPRESS ANDOR or AND keyword, any duplicate supertypes are removed from the list.

complex: a boolean that has the value TRUE if the **entity_definition** is the result of mapping ANDOR or AND supertypes in the application schema (see A.1.3); FALSE if the **entity_definition** is mapped directly from an entity type in the schema.

instantiable: a boolean that has the value FALSE if the entity type is declared as an ABSTRACT SUPERTYPE in the schema and TRUE otherwise.

independent: a boolean that has the value FALSE if the entity type is not independently instantiable because it is made available by a REFERENCE specification or is implicitly interfaced into the schema; TRUE if the entity type is declared locally within the schema or is made available by a USE specification.

attributes: the attributes that are declared or redeclared (see ISO 10303-11: 9.2.3.4) in the entity type. Attributes inherited from a supertype do not appear as elements of this set. As entities established by the mapping of EXPRESS ANDOR or AND keywords are considered subtypes of the constituent entity types, this set is empty for instances of **entity_definition** established by the EXPRESS ANDOR or AND supertype constraint.

uniqueness_rules: the uniqueness rules declared in the entity type. This set is empty for entities established by the mapping of EXPRESS ANDOR or AND supertype constraints.

global_rules: the global rules that name the entity type in the declaration of the rule.

6.4.13 attribute

An **attribute** is a property of an entity type. It may be explicit, inverse, or derived. An attribute has a name and a domain.

EXPRESS specification:

```
*)
ENTITY attribute
  ABSTRACT SUPERTYPE OF (ONEOF(derived_attribute, explicit_attribute,
   inverse_attribute));
  name             : express_id;
  parent_entity    : entity_definition;
END_ENTITY;
(*
```

Attribute definitions:

name: the name of the attribute.

parent_entity: the entity type in which the attribute is declared.

6.4.14 derived_attribute

A **derived_attribute** is an attribute whose value is computed by evaluating an expression. It may redeclare an explicit or a derived attribute (see ISO 10303-11: 9.2.3.4).

EXPRESS specification:

```
*)
ENTITY derived_attribute
  SUBTYPE OF (attribute);
  domain       : base_type;
  redeclaring : OPTIONAL explicit_or_derived;
END_ENTITY;
(*
```

Attribute definitions:

domain: the data type of the result of the attribute value derivation.

redeclaring: if present, the attribute being redeclared.

6.4.15 explicit_attribute

An **explicit_attribute** is an attribute whose domain is specified explicitly. It may redeclare an explicit attribute (see ISO 10303-11: 9.2.3.4).

EXPRESS specification:

```
*)
ENTITY explicit_attribute
  SUBTYPE OF (attribute);
  domain         : base_type;
  redeclaring    : OPTIONAL explicit_attribute;
  optional_flag : BOOLEAN;
END_ENTITY;
(*
```

Attribute definitions:

domain: the data type referenced by the attribute.

redeclaring: if present, the attribute being redeclared.

optional_flag: a boolean that has the value TRUE if the attribute is declared OPTIONAL; FALSE if the attribute is not declared as OPTIONAL.

6.4.16 inverse_attribute

An **inverse_attribute** is an attribute that captures, and may constrain, the reverse direction of a relationship established by an **explicit_attribute** and that represents an EXPRESS INVERSE attribute. It may redeclare an inverse attribute (see ISO 10303-11: 9.2.3.4). An inverse attribute may be represented by a single entity type or a SET or BAG of an entity type. The entity type with the **inverse_attribute** is described as the current entity type and the entity type with the **explicit_attribute** is described as the referencing entity type (see ISO 10303-11: 9.2.1.3).

EXPRESS specification:

```
*)
ENTITY inverse_attribute
  SUBTYPE OF (attribute);
  domain          : entity_definition;
  redeclaring     : OPTIONAL inverse_attribute;
  inverted_attr   : explicit_attribute;
  min_cardinality : OPTIONAL bound;
  max_cardinality : OPTIONAL bound;
  duplicates      : BOOLEAN;
END_ENTITY;
(*
```

Attribute definitions:

domain: the referencing entity type defining the forward relationship; the source of the relationship.

redeclaring: if present, the attribute being redeclared.

inverted_attr: the attribute in the referencing entity type whose relationship is being inverted.

min_cardinality: if present, the minimum number of references from the inverted attribute in instances of the referencing entity type when the inverse attribute is represented as a SET or BAG. If not present, the inverse attribute is represented by a single entity data type, not a SET or BAG.

max_cardinality: if present, the maximum number of references from the inverted attribute in instances of the referencing entity type. If not present, the SET or BAG representing the inverse attribute does not specify the maximum number of references or the inverse is represented by a single entity data type.

duplicates: a boolean that has the value TRUE if the inverse attribute is represented by a BAG; FALSE if the inverse attribute is represented by a SET or a single entity type.

6.4.17 uniqueness_rule

A **uniqueness_rule** is the representation of an EXPRESS UNIQUE rule. It specifies a combination of attributes that are required to be unique within instances of the **entity_definition** within which the rule is declared.

EXPRESS specification:

```
*)
ENTITY uniqueness_rule;
  label          : OPTIONAL express_id;
  attributes     : LIST [1:?] OF attribute;
  parent_entity : entity_definition;
END_ENTITY;
(*
```

Attribute definitions:

label: if present, the name of the uniqueness rule.

attributes: the list of attributes participating in the uniqueness rule.

parent_entity: the entity type within which the rule is declared.

6.4.18 where_rule

A **where_rule** is a constraint. It constrains a population and represents an EXPRESS WHERE rule. When defined in an **entity_definition** the values of attributes in the defining entity type are constrained. When defined in a **defined_type** the domain of the defining type is constrained. When defined in a **global_rule** the values of the attributes in or the existence of instances of entity types to which the global rule applies are constrained.

EXPRESS specification:

```
*)
ENTITY where_rule;
  label       : OPTIONAL express_id;
  parent_item : type_or_rule;
END_ENTITY;
(*
```

Attribute definitions:

label: if present, the name of the where rule.

parent_item: the entity type, defined type or global rule in which the where rule is declared.

6.4.19 global_rule

A **global_rule** is a constraint. It constrains all instances of an entity type or constrains instances of multiple entity types and represents an EXPRESS RULE.

EXPRESS specification:

```
*)
ENTITY global_rule;
  name          : express_id;
  entities      : LIST [1:?] OF entity_definition;
  where_rules   : LIST [1:?] OF where_rule;
```

```
  parent_schema : schema_definition;
END_ENTITY;
(*
```

Attribute definitions:

name: the name of the rule.

entities: the entity types constrained by the rule as defined in the declaration of the rule in the order in which they appear in the declaration of the rule. Entities established by the EXPRESS ANDOR or AND supertype constraint shall not appear as elements of this list.

where_rules: the domain rules declared in the EXPRESS RULE in the order in which they appear in the RULE declaration.

parent_schema: the schema in which the rule is declared.

6.4.20 simple_type

A **simple_type** is an EXPRESS unstructured, built-in base type.

EXPRESS specification:

```
*)
ENTITY simple_type
  ABSTRACT SUPERTYPE OF (ONEOF(integer_type, real_type, string_type,
   binary_type, logical_type, boolean_type, number_type));
END_ENTITY;
(*
```

6.4.21 number_type

A **number_type** is a **simple_type** that represents the EXPRESS NUMBER type.

EXPRESS specification:

```
*)
ENTITY number_type
  SUBTYPE OF (simple_type);
END_ENTITY;
(*
```

6.4.22 integer_type

An **integer_type** is a **simple_type** that represents the EXPRESS INTEGER type.

EXPRESS specification:

```
*)
ENTITY integer_type
  SUBTYPE OF (simple_type);
END_ENTITY;
(*
```

6.4.23 real_type

A **real_type** is a **simple_type** that represents the EXPRESS REAL type. The minimum number of significant digits in a value of the REAL type may be specified.

EXPRESS specification:

```
*)
ENTITY real_type
  SUBTYPE OF (simple_type);
  precision : OPTIONAL bound;
END_ENTITY;
(*
```

Attribute definitions:

precision: if present, the minimum number of significant digits in a value of the type.

Informal propositions:

precision_positive: If specified, the precision bound value shall evaluate to a positive integer.

6.4.24 string_type

A **string_type** is a **simple_type** that represents the EXPRESS STRING type. A STRING is of either fixed or variable width. The width of a STRING may be specified.

EXPRESS specification:

```
*)
ENTITY string_type
  SUBTYPE OF (simple_type);
  width       : OPTIONAL bound;
  fixed_width : BOOLEAN;
END_ENTITY;
(*
```

Attribute definitions:

width: if present, the maximum, or for fixed width strings the exact, number of characters in a value of the type.

fixed_width: a boolean that has the value TRUE if the type has as its domain fixed width strings; FALSE if the type has as its domain variable width strings.

Informal propositions:

width_positive: If specified, the string width bound value shall evaluate to a positive integer.

6.4.25 binary_type

A **binary_type** is a **simple_type** that represents the EXPRESS BINARY type. A BINARY is of either fixed or variable width. The width of a BINARY may be specified.

EXPRESS specification:

```
*)
ENTITY binary_type
  SUBTYPE OF (simple_type);
  width       : OPTIONAL bound;
  fixed_width : BOOLEAN;
END_ENTITY;
(*
```

Attribute definitions:

width: if present, the maximum, or for fixed width binary types the exact, number of bits in a value of the type.

fixed_width: a boolean that has the value TRUE if the type has as its domain fixed width binary types; FALSE if the type has as its domain variable width binary types.

Informal propositions:

width_positive: If specified, the binary width bound value shall evaluate to a positive integer.

6.4.26 logical_type

A **logical_type** is a **simple_type** that represents the EXPRESS LOGICAL type.

EXPRESS specification:

```
*)
ENTITY logical_type
  SUBTYPE OF (simple_type);
END_ENTITY;
(*
```

6.4.27 boolean_type

A **boolean_type** is a **simple_type** that represents the EXPRESS BOOLEAN type.

EXPRESS specification:

```
*)
ENTITY boolean_type
  SUBTYPE OF (simple_type);
END_ENTITY;
(*
```

6.4.28 enumeration_type

An **enumeration_type** represents the EXPRESS ENUMERATION type.

EXPRESS specification:

```
*)
ENTITY enumeration_type;
  elements : LIST [1:?] OF UNIQUE express_id;
END_ENTITY;
(*
```

Attribute definitions:

elements: the list of the types values in the order in which they appear in the EXPRESS declaration of the enumeration.

6.4.29 select_type

A **select_type** represents the EXPRESS SELECT type.

EXPRESS specification:

```
*)
ENTITY select_type;
  selections : SET [1:?] OF named_type;
END_ENTITY;
(*
```

Attribute definitions:

selections: the set of selectable types.

6.4.30 aggregation_type

An **aggregation_type** is an EXPRESS data type whose values are collections of other values of a given base type.

EXPRESS specification:

```
*)
ENTITY aggregation_type
  ABSTRACT SUPERTYPE OF (ONEOF(variable_size_aggregation_type,
  array_type));
  element_type : base_type;
END_ENTITY;
(*
```

Attribute definitions:

element_type: the type of the elements that are contained in values of the aggregation type.

6.4.31 variable_size_aggregation_type

A **variable_size_aggregation_type** is an **aggregation_type** that may be declared as having a variable number of elements. The number of elements is bounded from below and may be bounded from above.

EXPRESS specification:

```
*)
ENTITY variable_size_aggregation_type
  ABSTRACT SUPERTYPE OF (ONEOF(set_type, bag_type, list_type))
  SUBTYPE OF (aggregation_type);
  lower_bound : bound;
  upper_bound : OPTIONAL bound;
END_ENTITY;
(*
```

Attribute definitions:

lower_bound: the minimum number of elements that can be in an instance of the type. The lower bound value is zero if no numeric expression is specified for its value in the schema within which the aggregation is declared.

upper_bound: if present, the maximum number of elements that can be in an instance of the type. If not present, the number of elements that can be in an instance of the type is not bounded from above.

Informal propositions:

valid_boundaries: The lower bound value shall not be greater than the upper bound value.

6.4.32 set_type

A **set_type** is a **variable_size_aggregation_type** that represents the EXPRESS SET type.

EXPRESS specification:

```
*)
ENTITY set_type
  SUBTYPE OF (variable_size_aggregation_type);
END_ENTITY;
(*
```

6.4.33 bag_type

A **bag_type** is a **variable_size_aggregation_type** that represents the EXPRESS BAG type.

EXPRESS specification:

```
*)
ENTITY bag_type
  SUBTYPE OF (variable_size_aggregation_type);
END_ENTITY;
(*
```

6.4.34 list_type

A **list_type** is a **variable_size_aggregation_type** that represents the EXPRESS LIST type. The elements of a list may be required to be unique.

EXPRESS specification:

```
*)
ENTITY list_type
  SUBTYPE OF (variable_size_aggregation_type);
  unique_flag : BOOLEAN;
END_ENTITY;
(*
```

Attribute definitions:

unique_flag: a boolean that has the value TRUE if the UNIQUE keyword was specified in the definition of the **list_type** in the schema; otherwise the boolean has the value FALSE.

6.4.35 array_type

An **array_type** is an **aggregation_type** that represents the EXPRESS ARRAY type. An array has lower and upper index values. The elements of an array are required to be unique if the UNIQUE keyword was specified. An array may contain indeterminate values at one or more index positions if the OPTIONAL keyword was specified.

EXPRESS specification:

```
*)
ENTITY array_type
  SUBTYPE OF (aggregation_type);
  lower_index    : bound;
  upper_index    : bound;
  unique_flag    : BOOLEAN;
  optional_flag : BOOLEAN;
END_ENTITY;
(*
```

Attribute definitions:

lower_index: the lowest valid index for instances of the type.

upper_index: the highest valid index for instances of the type.

unique_flag: a boolean that has the value TRUE if the UNIQUE keyword was specified in the definition of the **array_type** in the schema; otherwise the boolean has the value FALSE.

optional_flag: a boolean that has the value TRUE if the OPTIONAL keyword was specified in the definition of the array_type in the schema; otherwise the boolean has the value FALSE.

Informal propositions:

valid_boundaries: The lower index bound value shall not be greater than the upper index bound value.

6.4.36 bound

A **bound** is a limit on an EXPRESS aggregation, binary, string or real type specified as an integer-valued numeric expression. The value of the **bound** may be based solely on the schema within which it is declared or may depend upon a population of that schema.

EXPRESS specification:

```
*)
ENTITY bound
  ABSTRACT SUPERTYPE OF (ONEOF(integer_bound, population_dependent_bound));
END_ENTITY;
(*
```

6.4.37 population_dependent_bound

A **population_dependent_bound** is a **bound** whose value depends on a population of the schema within which it is declared.

EXPRESS specification:

```
*)
ENTITY population_dependent_bound
  SUBTYPE OF (bound);
END_ENTITY;
(*
```

6.4.38 integer_bound

An **integer_bound** is a **bound** whose value is based solely on the schema within which it is declared.

EXPRESS specification:

```
*)
ENTITY integer_bound
  SUBTYPE OF (bound);
  bound_value : INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

bound_value: the integer value for the bound.

```
*)
END_SCHEMA; -- SDAI_dictionary_schema
(*
```

7 SDAI session schema

The following EXPRESS declaration begins the SDAI session schema and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA SDAI_session_schema;

REFERENCE FROM SDAI_parameter_data_schema
  ( entity_instance,
  aggregate_instance );

USE FROM SDAI_population_schema;
(*
```

NOTE - The schemas referenced above can be found in the following clauses of this part of ISO 10303:

SDAI_parameter_data_schema	clause 9
SDAI_population_schema	clause 8

7.1 Introduction

The SDAI session schema defines the structure of the data required to manage an SDAI session. The current state of an SDAI session and its interaction with the SDAI implementation such as access modes, transactions, repositories, and session errors are made available by a population of the SDAI session schema. As with all schemas available in the data dictionary, the items interfaced into the SDAI session schema from the SDAI population schema, SDAI parameter data schema and SDAI dictionary schema through the SDAI population schema shall be resolved into the SDAI session schema (see A.1.1).

The SDAI session schema describes a single application, single user view of information about an SDAI session. The SDAI implementation shall make the instances of SDAI session schema entity data types not interfaced from another schema available in one SDAI-model. That SDAI-model shall be associated with one schema instance. Both the SDAI-model and schema instance shall be based upon the SDAI session schema. The **sdai_model.name** of the SDAI-model shall be 'SDAI_SESSION_SCHEMA_DATA'. The **schema_instance.name** of the schema instance shall be 'SDAI_SESSION_SCHEMA_INSTANCE'. The SDAI-model and schema instance need not persist beyond the end of the session.

Instances of SDAI session schema entities are created and modified only as the consequence of specific SDAI operations, not the entity or application instance operations.

EXAMPLE 6 - The Open Session operation creates instances of the session entity type and the implementation entity type.

7.2 Fundamental concepts and assumptions

The structure of the entities and attributes of this schema takes implementation and efficiency into account.

7.3 SDAI session schema type definitions

This subclause contains the session concepts defined using the EXPRESS TYPE construct.

7.3.1 access_type

An **access_type** specifies either read-only or read-write access for an **sdai_transaction** or **sdai_model**.

EXPRESS specification:

```
*)
TYPE access_type = ENUMERATION OF
  (read_only,
  read_write);
END_TYPE;
(*
```

Enumeration items:

read_only: the value indicating RO access is available.

read_write: the value indicating RW access is available.

7.3.2 error_base

An **error_base** is a selection of an **entity_instance** or an **aggregate_instance** and is related to a particular error generated by the SDAI implementation.

EXPRESS specification:

```
*)
TYPE error_base = SELECT
  (entity_instance,
  aggregate_instance);
END_TYPE;
(*
```

7.3.3 time_stamp

A **time_stamp** is date and time specification. The contents of the string shall correspond to the extended format for the complete calendar data as specified in 5.2.1.1 of ISO 8601 concatenated to the extended format for the time of day as specified either in 5.3.1.1 or in 5.3.3 of ISO 8601. The date and time shall be separated by the capital letter T as specified in 5.4.1 of ISO 8601. The alternate formats of 5.3.1.1 and 5.3.3 permit the optional inclusion of a time zone specifier.

EXPRESS specification:

```
*)
TYPE time_stamp = STRING(256);
END_TYPE;
(*
```

7.4 SDAI session schema entity definitions

This subclause contains the session concepts defined using the EXPRESS ENTITY construct.

7.4.1 sdai_session

An **sdai_session** represents the information describing an SDAI session at a point in time while the SDAI implementation is active. It contains information reflecting the state of the session with respect to transactions, errors, event recording, repositories and the data dictionary.

EXPRESS specification:

```
*)
ENTITY sdai_session;
  sdai_implementation : implementation;
  recording_active    : BOOLEAN;
  errors              : LIST [0:?] OF error_event;
  known_servers       : SET [1:?] OF sdai_repository;
  active_servers      : SET [1:?] OF sdai_repository;
  active_models       : SET [1:?] OF sdai_model;
  data_dictionary     : OPTIONAL schema_instance;
INVERSE
  active_transaction : SET [0:1] OF sdai_transaction FOR owning_session;
END_ENTITY;
(*
```

Attribute definitions:

sdai_implementation: the characteristics of the SDAI implementation.

recording_active: a boolean that has the value FALSE if the recording of events is inhibited; TRUE otherwise.

errors: the list of errors that have resulted from previously executed SDAI operations while event recording was active.

known_servers: the repositories available to the application in this session. The presence of particular repositories depends on the specific installation of an SDAI implementation.

active_servers: the repositories currently open in the session.

active_models: the **sdai_model**s currently being accessed in the session.

data_dictionary: if present, the schema instance, based upon the SDAI session schema, with which the SDAI-models constituting the data dictionary are associated. For SDAI implementations claiming conformance to implementation classes 2 through 6 this attribute shall have a value specified.

active_transaction: the transaction making access to SDAI-models and schema instances available in the current session.

7.4.2 implementation

An **implementation** represents a software product that provides the functionality defined by an SDAI language binding.

EXPRESS specification:

```
*)
ENTITY implementation;
  name                    : STRING;
  level                   : STRING;
  sdai_version            : STRING;
  binding_version         : STRING;
  implementation_class    : INTEGER;
  transaction_level       : INTEGER;
  expression_level        : INTEGER;
  recording_level         : INTEGER;
  scope_level             : INTEGER;
  domain_equivalence_level : INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

name: the name of **implementation** assigned by the implementor.

level: the software version level of the **implementation** assigned by the implementor.

sdai_version: the version of this part of ISO 10303 to which the implementation conforms. The value of this attribute shall follow the registration technique defined in ISO 10303-1: 4.3 and shall be the object identifier for the appropriate version of this part of ISO 10303 (see C.1).

binding_version: the version of the SDAI language binding supported as specified in the SDAI language binding.

implementation_class: the implementation class specified in this part of ISO 10303 to which the **implementation** conforms (see 13.2).

transaction_level: the level of transaction supported by the implementation (see 13.1.1).

expression_level: the level of expression evaluation supported by the implementation (see 13.1.2).

recording_level: the level of event recording supported by the implementation (see 13.1.3).

scope_level: the level of SCOPE supported by the implementation (see 13.1.4).

domain_equivalence_level: the level of domain equivalence supported by the implementation (see 13.1.5).

Informal propositions:

class1to7: implementation_class shall have a value from 1 to 7 where the value corresponds to an

implementation class specified in 13.2.

trans1to3: transaction_level shall have a value from 1 to 3 where the value corresponds to a level of transaction specified in 13.1.1.

expr1to4: expression_level shall have a value from 1 to 4 where the value corresponds to a level of expression evaluation specified in 13.1.2.

rec1to2: recording_level shall have a value from 1 to 2 where the value corresponds to a level of event recording specified in 13.1.3.

scope1to2: scope_level shall have a value from 1 to 2 where the value corresponds to a level of SCOPE support specified in 13.1.4.

equiv1to2: domain_equivalence_level shall have a value from 1 to 2 where the value corresponds to a level of domain equivalence support specified in 13.1.5.

7.4.3 sdai_repository

An **sdai_repository** represents the identification of a facility where **sdai_model**s and **schema_instance**s can be stored during a session.

NOTE - This is intended to support the physical location of the SDAI-models and schema instances.

EXPRESS specification:

```
*)
ENTITY sdai_repository;
  name         : STRING;
  contents     : sdai_repository_contents;
  description : STRING;
INVERSE
  session : sdai_session FOR known_servers;
UNIQUE
  UR1: name, session;
END_ENTITY;
(*
```

Attribute definitions:

name: the name of the **sdai_repository**. The name is case sensitive.

contents: the SDAI-models and schema instances that exist in the repository.

description: a description of the repository.

session: the current session.

Formal propositions:

UR1: the name shall be unique within the current session.

7.4.4 sdai_repository_contents

An **sdai_repository_contents** identifies the **sdai_models** and **schema_instances** that exist within a repository.

EXPRESS specification:

```
*)
ENTITY sdai_repository_contents;
  models   : SET [0:?] OF sdai_model;
  schemas  : SET [0:?] OF schema_instance;
INVERSE
  repository : sdai_repository FOR contents;
END_ENTITY;
(*
```

Attribute definitions:

models: the SDAI-models in the repository.

schemas: the schema instances in the repository.

repository: the repository containing the SDAI-models and schema instances.

7.4.5 sdai_transaction

An **sdai_transaction** describes the currently available access to data (RW or RO) within an **sdai_session**. Transactions shall not exist outside of a session and only one transaction shall be active at any given time. Transactions are required only of SDAI implementations supporting transaction level 3 (see 13.1.1).

EXPRESS specification:

```
*)
ENTITY sdai_transaction;
  mode          : access_type;
  owning_session : sdai_session;
END_ENTITY;
(*
```

Attribute definitions:

mode: the read-only or read-write access provided by the transaction within the **sdai_session**.

owning_session: the **sdai_session** within which the transaction is active.

7.4.6 event

An **event** is a notation of some occurrence related to an SDAI operation at some moment during a session.

NOTE - Future editions of SDAI may expand the types of events that may be recorded beyond the error recording specified in this edition of this part of ISO 10303.

EXPRESS specification:

```
*)
ENTITY event
  ABSTRACT SUPERTYPE OF (error_event);
  function_id : STRING;
  time        : time_stamp;
END_ENTITY;
(*
```

Attribute definitions:

function_id: the identifier for the SDAI function or subroutine with which the event is associated. The values of this attribute and their correspondence to SDAI operations are specified in the SDAI language binding.

time: the time stamp indicating when the event occurred.

7.4.7 error_event

An **error_event** is an **event** generated as result of an SDAI operation that has failed to execute successfully or as the result of the Record error operation.

EXPRESS specification:

```
*)
ENTITY error_event
  SUBTYPE OF (event);
  error       : INTEGER;
  description : OPTIONAL STRING;
  base        : OPTIONAL error_base;
END_ENTITY;
(*
```

Attribute definitions:

error: the error code for the error (see clause 11).

description: if present, the description of the error that has occurred.

base: if present, the instance related to or causing the error (see clause 11).

```
*)
END_SCHEMA; -- SDAI_session_schema
(*
```

8 SDAI population schema

The following EXPRESS declaration begins the SDAI population schema and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA SDAI_population_schema;

REFERENCE FROM SDAI_parameter_data_schema
  ( entity_instance,
    application_instance );

REFERENCE FROM SDAI_session_schema
  ( sdai_session,
    sdai_repository,
    access_type,
    time_stamp );
(*
```

In the case where the implementation claims conformance to implementation classes 2 through 6 or where the implementation supports access to the SDAI data dictionary, the following EXPRESS declaration shall be included in the SDAI population schema. Otherwise the EXPRESS declarations in 8.3 shall be included in the SDAI population schema.

EXPRESS specification:

```
*)
USE FROM SDAI_dictionary_schema;
(*
```

NOTE - The schemas referenced above can be found in the following clauses of this part of ISO 10303:

SDAI_parameter_data_schema	clause 9
SDAI_session_schema	clause 7
SDAI_dictionary_schema	clause 6

8.1 Introduction

The SDAI population schema defines a structure for the organization, creation and management of instances of EXPRESS entities. This schema is interfaced by the SDAI session schema and all items in this schema are resolved into the SDAI session schema in the population of the SDAI dictionary schema (see A.1.1).

Implementations that do not provide access to a data dictionary shall utilize the SDAI population schema without the SDAI dictionary schema being interfaced. The references to items defined in the SDAI dictionary schema are replaced by string values.

The instances of entity data types defined in the SDAI population schema provide a management structure for entity instances of both application schemas and the SDAI dictionary schema. This management structure need not exist for the instances of the entity data types defined in the SDAI population schema

itself (e.g., the instances of **sdai_model** need not exist within an **sdai_model**). The mechanism for making available the instances of entity types declared in the SDAI population schema is left to the implementation.

8.2 Fundamental concepts and assumptions

The structure of the entities and attributes of this schema takes implementation and efficiency into account.

8.3 SDAI population schema type definitions

This subclause contains the population concepts defined using the EXPRESS TYPE construct. In the case where the implementation does not support access to a data dictionary the EXPRESS declarations in this subclause shall be included in the SDAI population schema. Otherwise, these EXPRESS declarations shall not be included in the SDAI population schema.

8.3.1 schema_definition

A **schema_definition** represents the same concept as defined in and is represented by the **schema_definition.name** in 6.4.1.

EXPRESS specification:

```
*)
TYPE schema_definition = STRING;
END_TYPE;
(*
```

8.3.2 entity_definition

A **entity_definition** represents the same concept as defined in and is represented by the **entity_definition.name** in 6.4.12.

EXPRESS specification:

```
*)
TYPE entity_definition = STRING;
END_TYPE;
(*
```

8.4 SDAI population schema entity definitions

This subclause contains the population concepts defined using the EXPRESS ENTITY construct.

8.4.1 schema_instance

A **schema_instance** is a logical collection of **sdai_models**. It is used as the domain for global rule validation, as the domain over which references between entity instances in different SDAI-models are supported and as the domain for uniqueness validation. A **schema_instance** is based upon one schema. Associating SDAI-models with the **schema_instance** shall be supported when the SDAI-model is based

upon the same schema as the **schema_instance**. Associating SDAI-models with the **schema_instance** when the SDAI-model is based upon another schema shall also be supported provided the schema upon which the SDAI-model is based contains constructs declared as domain equivalent with constructs in the schema upon which the **schema_instance** is based. Although a **schema_instance** exists in one repository, associating SDAI-models from any repository with the **schema_instance** shall be supported.

EXPRESS specification:

```
*)
ENTITY schema_instance;
  name              : STRING;
  associated_models : SET [0:?] OF sdai_model;
  native_schema     : schema_definition;
  repository        : sdai_repository;
  change_date       : OPTIONAL time_stamp;
  validation_date   : time_stamp;
  validation_result : LOGICAL;
  validation_level  : INTEGER;
UNIQUE
  UR1: name, repository;
WHERE
  WR1: SELF IN SELF.repository.contents.schemas;
END_ENTITY;
(*
```

Attribute definitions:

name: the name of the **schema_instance**. The name is case sensitive.

associated_models: the SDAI-models associated with the schema instance.

native_schema: the schema upon which the schema instance is based.

repository: the repository within which the schema instance was created.

change_date: if present, the creation date or date of the most recent add or remove of an SDAI-model from the current schema instance.

validation_date: the date of the most recent Validate schema instance operation performed on the current schema instance.

validation_result: the result of the most recent Validate schema instance operation performed on the current schema instance.

validation_level: the level of expression evaluation for validation of the implementation that performed the most recent Validate schema instance operation on the current schema instance (see 13.1.2).

Formal propositions:

UR1: The name shall be unique within the repository containing the schema instance.

Informal propositions:

dictionary instance: During an SDAI session of an implementation supporting the SDAI data dictionary, there shall be an instance of **schema_instance** with which all SDAI-models constituting the SDAI data dictionary shall be associated.

8.4.2 sdai_model

An **sdai_model** is a grouping mechanism consisting of a set of related entity instances based upon a **schema_definition**.

NOTE - The relationship of the entity instances grouped into an SDAI-model is not specified. However, it is expected that the grouping is based upon some logical relationship between the entity instances that is useful to the application program in its management of application data.

EXPRESS specification:

```
*)
ENTITY sdai_model;
  name              : STRING;
  contents          : sdai_model_contents;
  underlying_schema : schema_definition;
  repository        : sdai_repository;
  change_date       : OPTIONAL time_stamp;
  mode              : OPTIONAL access_type;
INVERSE
  associated_with : SET [0:?] OF schema_instance FOR associated_models;
UNIQUE
  UR1: repository, name;
WHERE
  WR1: SELF IN SELF.repository.contents.models;
END_ENTITY;
(*
```

Attribute definitions:

name: the identifier for this **sdai_model**. The name is case sensitive.

contents: the collection mechanism for entity instances within the **sdai_model**.

underlying_schema: the schema that defines the structure of the data that appears in the SDAI-model.

repository: the repository within which the SDAI-model was created.

change_date: if present, the creation date or date of most recent modification, including creation or deletion, to an entity instance within the current SDAI-model.

mode: if present, the current access mode for the **sdai_model**. If not present, the **sdai_model** is not open.

associated_with: the schema instances with which the SDAI-model has been associated.

Formal propositions:

UR1: The name shall be unique within the repository containing the SDAI-model.

8.4.3 sdai_model_contents

An **sdai_model_contents** contains the entity instances making up an **sdai_model**. The entity instances are available in a single collection regardless of entity data type and grouped by entity data type into multiple collections.

NOTE - The availability and grouping of the entity instances in three alternative attributes is one case where ease of access for the application programmer is taken into account in this schema.

EXPRESS specification:

```
*)
ENTITY sdai_model_contents;
  instances          : SET [0:?] OF entity_instance;
  folders            : SET [0:?] OF entity_extent;
  populated_folders  : SET [0:?] OF entity_extent;
END_ENTITY;
(*
```

Attribute definitions:

instances: the set of all entity instances in the **sdai_model** regardless of entity data type.

folders: the set of **entity_extent**s for all entity types available in the schema corresponding to the SDAI-model. This set contains one member for each **entity_definition** found in the schema governing the SDAI-model regardless of whether any entity instances of that entity type currently exist.

populated_folders: a subset of **folders**, containing the set of **entity_extent**s for which instances currently exist in the SDAI-model.

Informal propositions:

IP1: The set **sdai_model_contents.instances** contains the same entity instances as the union of the set of extents **sdai_model_contents.populated_folders** contains.

8.4.4 entity_extent

An **entity_extent** groups all instances of an entity data type that exist in an **sdai_model**. This grouping includes instances of the specified **entity_definition**, instances of all subtypes of the **entity_definition** and instances of other **entity_definition**s resulting from the mapping of the EXPRESS AND and ANDOR constraints as described in annex A which contains the entity data type as a constituent.

NOTE - Extents of instances of a particular type are useful as entry points within an SDAI-model for gaining access to instances of any root entity types in schemas containing tree or hierarchical structures.

EXPRESS specification:

```
*)
ENTITY entity_extent;
  definition : entity_definition;
  instances  : SET [0:?] OF entity_instance;
INVERSE
  owned_by : sdai_model_contents FOR folders;
END_ENTITY;
(*
```

Attribute definitions:

definition: the **entity_definition** whose instances are contained in the folder.

instances: the entity instances contained in the folder.

owned_by: the SDAI-model contents owning the **entity_extent**.

8.4.5 scope

A **scope** provides the structure to support the scope of reference and existence relationships between entity instances defined by the ISO 10303-21 SCOPE construct (see ISO 10303-21: 10.3). ISO 10303-22 refers to the application instance within which the **scope** structure is defined as the owner and those application instances appearing within the **scope** structure as being owned by the **scope** owner. Application instances owned by a **scope** may themselves own a **scope**. A **scope** exists in the same SDAI-model as the **application_instance** owning the **scope** (e.g., if the SDAI-model containing the application instance is deleted, then the **scope** is deleted).

EXPRESS specification:

```
*)
ENTITY scope;
  owner       : application_instance;
  owned       : SET [1:?] OF application_instance;
  export_list : SET [0:?] OF application_instance;
UNIQUE
  UR1: owner;
WHERE
  WR2: NOT (owner IN owned);
END_ENTITY;
(*
```

Attribute definitions:

owner: the application instance within which the current **scope** is defined.

owned: the application instances defined to be within the current **scope**.

export_list: the application instances whose ability to be referenced has been extended by being exported from the current **scope**.

Formal propositions:

UR1: An application instance shall not be the owner of more than one **scope**.

WR2: The application instance shall not be included in the set of application instances for which it is the owner.

Informal propositions:

IP1: The SCOPE structure shall be acyclic.

IP2: Each exported application instance shall be a member of the set of owned application instances, or a member of the export list of one instance of **scope**, whose owner is a member of owned.

```
*)
END_SCHEMA; -- SDAI_population_schema
(*
```

9 SDAI parameter data schema

The following EXPRESS declaration begins the SDAI parameter data schema and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA SDAI_parameter_data_schema;

REFERENCE FROM SDAI_population_schema
  ( schema_instance,
    sdai_model,
    sdai_model_contents );

REFERENCE FROM SDAI_session_schema
  ( sdai_repository );
(*
```

NOTE - The schemas referenced above can be found in the following clauses of this part of ISO 10303:

SDAI_sessions_schema	clause 7
SDAI_population_schema	clause 8

9.1 Introduction

The SDAI parameter data schema provides conceptual descriptions of the data that is passed as parameters or manipulated through the SDAI. Unlike the SDAI dictionary schema, the SDAI session schema and the SDAI population schema, the SDAI parameter data schema need not be implemented nor is a population of this schema required as part of the SDAI data dictionary. This schema is defined to support the description of the SDAI operations and the definition of the SDAI environment in which entity instances exist. In several cases, this clause does not include an EXPRESS specification for the attributes of an entity

declared in the SDAI parameter data schema in normative text. Instead, an EXPRESS specification of these attributes is provided in an example with only the text description of the attribute in normative text in order to support the description of the parameter rather than specifying its implementation

The SDAI parameter data schema does describes the subtype relationships among the instances of entity data types manageable via the SDAI operations. These subtype relationships are used to categorize the SDAI operations. This categorization allows the specification of the behaviour of all SDAI implementations with respect to the types of entity instances that may be used as parameters or return values for SDAI operations.

SDAI language bindings specify data types for **entity_instance** and its subtypes. SDAI implementations shall provide data types for these entity types that behave as if the language implements the subtype relationships defined in this schema. The data type that represents the entity type **application_instance** shall behave as a supertype of every entity type declared in an application schema. The data type that represents the entity type **dictionary_instance** shall behave as a supertype of every entity type declared in the SDAI dictionary schema. The data type that represents the entity type **session_instance** shall behave as a supertype of every entity type declared in the SDAI session schema.

EXAMPLE 7 - Given application1 schema the entity data type e1 would have **application_instance** from the SDAI data type schema as an immediate supertype. Entity data type e2 would not need **application_instance** as an immediate supertype as it would inherit aspects via e1.

```
SCHEMA application1;
ENTITY e1;
END_ENTITY;
ENTITY e2
  SUBTYPE OF (e1);
END_ENTITY;
END_SCHEMA;
```

The subtype relationships defined in this subclause shall not appear in the population of the SDAI dictionary schema for any application schema, the SDAI dictionary schema or the SDAI session schema.

9.2 Fundamental concepts and assumptions

This schema assumes that a software system provides the representation of integer, real, character, bit and logical data types.

9.3 SDAI parameter data schema type definitions

This subclause contains the parameter data concepts defined using the EXPRESS TYPE construct.

9.3.1 primitive

A **primitive** is a selection of an **aggregate_instance** or an **assignable_primitive**. It is a value that may be the representation of an attribute of an EXPRESS entity instance.

EXPRESS specification:

```
*)
TYPE primitive = SELECT
  (aggregate_instance,
  assignable_primitive);
END_TYPE;
(*
```

9.3.2 assignable_primitive

An **assignable_primitive** is a selection of an **entity_instance**, a **string_value**, a **binary_value**, an **integer_value**, a **number_value**, an **enumeration_value**, a **select_value**, a **real_value**, a **boolean_value**, or a **logical_value**. It is a value that may be directly assigned to an attribute of an EXPRESS entity instance through an SDAI operation.

EXPRESS specification:

```
*)
TYPE assignable_primitive = SELECT
  (entity_instance,
  string_value,
  binary_value,
  integer_value,
  number_value,
  enumeration_value,
  select_value,
  real_value,
  boolean_value,
  logical_value);
END_TYPE;
(*
```

9.3.3 aggregate_primitive

An **aggregate_primitive** is a selection of an **aggregate_instance** or a **select_aggregate_instance**. It is a value whose underlying type is an aggregate instance.

EXPRESS specification:

```
*)
TYPE aggregate_primitive = SELECT
  (aggregate_instance,
  select_aggregate_instance);
END_TYPE;
(*
```

9.3.4 string_value

A **string_value** is the value associated with a data type that is defined by an EXPRESS STRING.

EXPRESS specification:

```
*)
TYPE string_value = STRING;
```

```
END_TYPE;
(*
```

9.3.5 binary_value

A **binary_value** is the value associated with a data type that is defined by an EXPRESS BINARY.

EXPRESS specification:

```
*)
TYPE binary_value = BINARY;
END_TYPE;
(*
```

9.3.6 integer_value

A **integer_value** is the value associated with a data type that is defined by an EXPRESS INTEGER.

EXPRESS specification:

```
*)
TYPE integer_value = INTEGER;
END_TYPE;
(*
```

9.3.7 real_value

A **real_value** is the value associated with a data type that is defined by an EXPRESS REAL.

EXPRESS specification:

```
*)
TYPE real_value = REAL;
END_TYPE;
(*
```

9.3.8 number_value

A **number_value** is the value associated with a data type that is defined by an EXPRESS NUMBER.

EXPRESS specification:

```
*)
TYPE number_value = SELECT
  (real_value,
  integer_value);
END_TYPE;
(*
```

9.3.9 boolean_value

A **boolean_value** is the value associated with a data type that is defined by an EXPRESS BOOLEAN

EXPRESS specification:

```
*)
TYPE boolean_value = BOOLEAN;
END_TYPE;
(*
```

9.3.10 logical_value

A **logical_value** is the value associated with a data type that is defined by an EXPRESS LOGICAL.

EXPRESS specification:

```
*)
TYPE logical_value = LOGICAL;
END_TYPE;
(*
```

9.3.11 bound_instance_value

A **bound_instance_value** is the value of **bound** that has a current value. The current value is either the **integer_bound.bound_value** or the value of the **population_dependent_bound** when the population of the schema declaring the **population_dependent_bound** is sufficient to evaluate the integer-valued expression specifying the value.

EXPRESS specification:

```
*)
TYPE bound_instance_value = INTEGER;
END_TYPE;
(*
```

9.3.12 query_source

A **query_source** is a selection of an **aggregate_instance**, an **sdai_model**, a **schema_instance** or an **sdai_repository**. It is the domain over which an SDAI query operation is executed.

EXPRESS specification:

```
*)
TYPE query_source = SELECT
  (aggregate_instance,
   sdai_model,
   sdai_repository,
   schema_instance) ;
END_TYPE;
(*
```

9.4 SDAI parameter data schema entity definitions

This subclause contains the parameter data concepts defined using the EXPRESS ENTITY construct.

9.4.1 iterator

An **iterator** is a mechanism by which SDAI operations refer to members of aggregates. More than one **iterator** may be defined on an aggregate. Each operates independently of the others; the behaviour of one iterator resulting from the modification of the aggregate using another iterator is not specified in this part of ISO 10303.

EXPRESS specification:

```
*)
ENTITY iterator;
  subject        : aggregate_instance;
  current_member : OPTIONAL primitive;
END_ENTITY;
(*
```

Attribute definitions:

subject: the aggregate accessed by the **iterator**.

current_member: if present, the current aggregation member referenced by the **iterator**. An **iterator** references no current member if it is at the beginning of an aggregate, at the end of an aggregate or at an array element whose value is unset.

9.4.2 entity_instance

An **entity_instance** is an instance based upon an **entity_definition** (see 6.4.12). This entity type is the supertype of all entity types whose instances may be queried or manipulated through an SDAI implementation. The attributes of an **entity_instance** are described in text but no EXPRESS specification is provided as the attributes need not be implemented. The relationship between **entity_instance** and its subtypes shall be supported in all SDAI implementations.

EXPRESS specification:

```
*)
ENTITY entity_instance
  ABSTRACT SUPERTYPE OF (ONEOF(sdai_instance, application_instance));
END_ENTITY;
(*
```

EXAMPLE 8 - The following describes a possible EXPRESS specification of the capabilities required of an entity instance.

```
ENTITY entity_instance
ABSTRACT SUPERTYPE OF (ONEOF(sdai_instance, application_instance));
  owning_model : sdai_model;
  definition   : entity_definition;
  values       : LIST [0:?] OF attribute_value;
```

```
WHERE
  WR1: SELF IN SELF\owning_model.contents.instances;
END_ENTITY;
```

Attribute definitions:

owning_model: the **sdai_model** to which this **entity_instance** belongs.

definition: the entity type of which this is an instance.

values: the values of the explicit attributes of this **entity_instance**.

Formal propositions:

WR1: The **entity_instance** shall be a member of the **sdai_model_contents.instances** describing the contents of the **sdai_model** within which the **entity_instance** exists.

9.4.3 application_instance

An **application_instance** is an **entity_instance** whose definition is an entity type defined in or interfaced into an application schema. The attributes of an **application_instance** are described in text but no EXPRESS specification is provided as the attributes need not be implemented. The subtype relationship between **application_instance** and **entity_instance** shall be supported in all SDAI implementations.

EXPRESS specification:

```
*)
ENTITY application_instance
  SUBTYPE OF (entity_instance);
END_ENTITY;
(*
```

EXAMPLE 9 - The following describes a possible EXPRESS specification of the capabilities required of an application instance.

```
ENTITY application_instance
  SUBTYPE OF (entity_instance);
  persistent_label : OPTIONAL STRING;
END_ENTITY;
```

Attribute definitions:

persistent_label: if present, the name associated with the application instance (see 10.11.6).

9.4.4 sdai_instance

An **sdai_instance** is an **entity_instance** whose definition is an entity type defined in an SDAI schema. The subtype relationship between **sdai_instance** and **entity_instance** and the subtype relationships between **dictionary_instance** and **session_instance** and **sdai_instance** shall be supported in all SDAI implementations.

EXPRESS specification:

```
*)
ENTITY sdai_instance
  ABSTRACT SUPERTYPE OF (ONEOF(dictionary_instance, session_instance))
  SUBTYPE OF (entity_instance);
END_ENTITY;
(*
```

9.4.5 dictionary_instance

A **dictionary_instance** is an **sdai_instance** whose definition is an entity type defined in the SDAI dictionary schema (see clause 6). The subtype relationship between **dictionary_instance** and **sdai_instance** shall be supported in all SDAI implementations.

EXPRESS specification:

```
*)
ENTITY dictionary_instance
  SUBTYPE OF (sdai_instance);
END_ENTITY;
(*
```

9.4.6 session_instance

A **session_instance** is an **sdai_instance** whose definition is an entity type defined in the SDAI session schema or interfaced into the SDAI session schema from the SDAI population schema (see clauses 7 and 8). Entity types interfaced into the SDAI session schema from the SDAI dictionary schema are not instances of **session_instance**. The subtype relationship between **session_instance** and **sdai_instance** shall be supported in all SDAI implementations.

EXPRESS specification:

```
*)
ENTITY session_instance
  SUBTYPE OF (sdai_instance);
END_ENTITY;
(*
```

9.4.7 attribute_value

An **attribute_value** is the value associated with an attribute of an **entity_instance**. The attributes of an **attribute_value** are described in text but no EXPRESS specification is provided as the attributes need not be implemented.

EXPRESS specification:

```
*)
ENTITY attribute_value;
END_ENTITY;
(*
```

EXAMPLE 10 - The following describes a possible EXPRESS specification of the capabilities required of an attribute value.

```
ENTITY attribute_value;
  data_value           : OPTIONAL primitive;
  attribute_definition : attribute;
DERIVE
  value_set : BOOLEAN := EXISTS(data_value);
END_ENTITY;
```

Attribute definitions:

data_value: if present, the value associated with the attribute; if not present, the attribute has no representation.

attribute_definition: the dictionary schema definition of the **attribute** for which this is a value.

value_set: a boolean that has the value TRUE if the attribute has a value; FALSE if the value of the attribute is not set.

9.4.8 select_value

A **select_value** is the value associated with a data type that is defined by an EXPRESS TYPE that is a SELECT. A **select_value** maintains information sufficient to uniquely identify which of the possible paths it instantiates in the case where the schema defines graphs of TYPEs, possibly including SELECT types, and, without additional information beyond the type, the instantiation may be ambiguous. The attributes of a **select_value** are described in text but no EXPRESS specification is provided as the attributes need not be implemented.

EXPRESS specification:

```
*)
ENTITY select_value
  SUPERTYPE OF (ONEOF(select_aggregate_instance));
END_ENTITY;
(*
```

EXAMPLE 11 - The following describes a possible EXPRESS specification of the capabilities required of a select value.

```
ENTITY select_value
  SUPERTYPE OF (ONEOF(select_aggregate_instance));
  data_value : OPTIONAL primitive;
  data_type  : LIST [0:?] OF defined_type;
END_ENTITY;
```

Attribute definitions:

data_value: the value associated with the data type.

data_type: the types sufficient to identify the instantiation of the value in the order in which they were provided by the application programmer.

9.4.9 select_aggregate_instance

A **select_aggregate_instance** is a **select_value** that has as its **data_value** an aggregate instance.

EXPRESS specification:

```
*)
ENTITY select_aggregate_instance
  SUBTYPE OF (select_value);
END_ENTITY;
(*
```

Informal propositions:

IP1: The **data_value** shall be an aggregate instance.

9.4.10 enumeration_value

An **enumeration_value** is the value associated with a data type that is an EXPRESS ENUMERATION. The attributes of an **enumeration_value** are described in text but no EXPRESS specification is provided as the attributes need not be implemented.

EXPRESS specification:

```
*)
ENTITY enumeration_value;
END_ENTITY;
(*
```

EXAMPLE 12 - The following describes a possible EXPRESS specification of the capabilities required of an enumeration value.

```
ENTITY enumeration_value;
  enumeration_name     : INTEGER;
  enumeration_typename : defined_type;
END_ENTITY;
```

Attribute definitions:

enumeration_name: the position within the **enumeration_type.elements** list corresponding to the enumeration item that is the value associated with the data type (see 6.4.28).

enumeration_typename: the **defined_type** defining the enumeration item that is the value of the data type.

Informal propositions:

IP1: the **defined_type** referenced in **enumeration_typename** shall be resolvable through the TYPEs in the schema such that it may have as its **defined_type.domain** an **enumeration_type**.

9.4.11 aggregate_instance

An **aggregate_instance** is an instance of an **aggregation_type**.

EXPRESS specification:

```
*)
ENTITY aggregate_instance
  ABSTRACT SUPERTYPE OF (ONEOF(unordered_collection, ordered_collection));
END_ENTITY;
(*
```

9.4.12 unordered_collection

An **unordered_collection** is an **aggregate_instance** that is an instance of a bag or set type.

EXPRESS specification:

```
*)
ENTITY unordered_collection
  ABSTRACT SUPERTYPE OF (ONEOF(set_instance, bag_instance))
  SUBTYPE OF (aggregate_instance);

END_ENTITY;
(*
```

9.4.13 set_instance

A **set_instance** is an **unordered_collection** that is an instance of a set type. The attributes of a **set_instance** are described in text but no EXPRESS specification is provided as the attributes need not be implemented.

EXPRESS specification:

```
*)
ENTITY set_instance
  SUBTYPE OF (unordered_collection);
END_ENTITY;
(*
```

EXAMPLE 13 - The following describes a possible EXPRESS specification of the capabilities required of a set instance.

```
ENTITY set_instance
  SUBTYPE OF (unordered_collection);
  set_definition : set_type;
  contents       : SET[0:?] OF primitive;
END_ENTITY;
```

Attribute definitions:

set_definition: the dictionary schema definition of the set.

contents: the values in the set.

9.4.14 bag_instance

A **bag_instance** is an **unordered_collection** that is an instance of a bag type. The attributes of a **bag_instance** are described in text but no EXPRESS specification is provided as the attributes need not be implemented.

EXPRESS specification:

```
*)
ENTITY bag_instance
  SUBTYPE OF (unordered_collection);
END_ENTITY;
(*
```

EXAMPLE 14 - The following describes a possible EXPRESS specification of the capabilities required of a bag instance.

```
ENTITY bag_instance
  SUBTYPE OF (unordered_collection);
  bag_definition : bag_type;
  contents       : BAG [0:?] OF primitive;
END_ENTITY;
```

Attribute definitions:

bag_definition: the dictionary schema definition of the bag.

contents: the values in the bag.

9.4.15 ordered_collection

An **ordered_collection** is an **aggregate_instance** that is an instance of a list or array type.

EXPRESS specification:

```
*)
ENTITY ordered_collection
  ABSTRACT SUPERTYPE OF (ONEOF(list_instance, array_instance))
  SUBTYPE OF (aggregate_instance);
END_ENTITY;
(*
```

9.4.16 list_instance

A **list_instance** is an **ordered_collection** that is an instance of a list type.

EXPRESS specification:

```
*)
ENTITY list_instance
  ABSTRACT SUPERTYPE OF (ONEOF(non_persistent_list_instance,
  schema_defined_list_instance))
  SUBTYPE OF (ordered_collection);
END_ENTITY;
(*
```

9.4.17 schema_defined_list_instance

A **schema_defined_list_instance** is a **list_instance** that is an instance of a list type defined in an SDAI or application schema. The attributes of a **schema_defined_list_instance** are described in text but no EXPRESS specification is provided as the attributes need not be implemented.

EXPRESS specification:

```
*)
ENTITY schema_defined_list_instance
  SUBTYPE OF (list_instance);
END_ENTITY;
(*
```

EXAMPLE 15 - The following describes a possible EXPRESS specification of the capabilities required of a schema defined list instance.

```
ENTITY schema_defined_list_instance
  SUBTYPE OF (list_instance);
  list_definition : list_type;
  contents        : LIST[0:?] OF primitive;
END_ENTITY;
```

Attribute definitions:

list_definition: the dictionary schema definition of the list.

contents: the values in the list.

9.4.18 non_persistent_list_instance

A **non_persistent_list_instance** is a **list_instance** that is an instance of a non-persistent, unbounded list of **entity_instance**s.

EXPRESS specification:

```
*)
ENTITY non_persistent_list_instance
  SUBTYPE OF (list_instance);
END_ENTITY;
(*
```

EXAMPLE 16 - The following describes a possible EXPRESS specification of the capabilities required of a non-persistent list instance.

```
ENTITY non_persistent_list_instance
  SUBTYPE OF (list_instance);
  contents : LIST[0:?] OF entity_instance;
END_ENTITY;
```

Attribute definitions:

contents: the values in the list.

9.4.19 array_instance

An **array_instance** is an **ordered_collection** that is an instance of an array type. The valid index values of an **array_instance** are set when the **array_instance** is created. Subsequent array operations may reset the index values for an **array_instance** whose index values depend upon a population of the schema in which the array type is defined (see 10.18.3). The attributes of an **array_instance** are described in text but no EXPRESS specification is provided as the attributes need not be implemented.

EXPRESS specification:

```
*)
ENTITY array_instance
  SUPERTYPE OF (ONEOF(application_indexed_array_instance))
  SUBTYPE OF (ordered_collection);
END_ENTITY;
(*
```

Attribute definitions:

array_definition: the dictionary schema definition of the array.

contents: the values of **primitive** that are the elements of the array. The indices of the array are specified as **bound_instance_values**.

9.4.20 application_indexed_array_instance

An **application_indexed_array_instance** is an **array_instance** that is an instance of an array type whose upper and lower index values are set by the application when the array instance is created. The Reset array index operation may reset the valid index positions for an **application_indexed-_array_instance** (see 10.18.4).

EXPRESS specification:

```
*)
ENTITY application_indexed_array_instance
  SUBTYPE OF (array_instance);
END_ENTITY;
(*

*)
END_SCHEMA; -- SDAI_parameter_data_schema
(*
```

10 SDAI operations

10.1 Introduction

This clause defines the SDAI operations. This clause does not specify how these operations are provided nor how specific inputs, outputs and possible error indicators are handled by any SDAI language binding.

NOTE 1 - A particular operation may map into one or more functions in some binding and these functions may have one or more input, output or error parameters conveyed implicitly, e.g., by the name of the function in an early binding.

Each operation is described by the following as required:

— a description of the function or the service performed by the operation;

— Input: if required, the information required to be specified prior to execution of the operation. Each input parameter is specified with a name with each word making up the name beginning with an uppercase character, the type of the parameter according to the schemas in clauses 6 through 9 with the bold face, lower case words identifying the type name and by a text description;

— Output: if required, the information made available to the application after the successful execution of the operation. The output parameters are specified using the same conventions as the input parameters;

NOTE 2 - Some parameters may be both Input and Output.

— Possible error indicators: the conditions that result in the operation terminating unsuccessfully. The text describing the error indicator may differ from that found in clause 11 in order to more specifically describe the type of the parameter associated with the error indicator for an operation;

— Effect on SDAI environment: if required, a description of how the input parameters and instances of entities from the schemas in clauses 6 through 9 are affected by the operation. In many cases the attribute of a particular entity instance is referred to with the syntax **Parameter.attribute** where **Parameter** is the input or output parameter name and **attribute** is the attribute of the entity instance for **Parameter**.

10.2 Fundamental concepts and assumptions

The following concepts and assumptions apply:

— the characteristics of the parameters of the SDAI operations are described in the SDAI dictionary schema, the SDAI session schema and the SDAI population schema. The subtype relationships between types of entity instances is described in the SDAI parameter data schema;

— entity instance operations are applicable to all instances of any entity data type defined in any SDAI or application schema available to the application;

— application instance operations are applicable only to instances of entity data types defined in application schemas available to the application;

— when an operation needs to access an entity instance in an SDAI-model that is not active in the current session and the repository containing that SDAI-model is open, function equivalent to the Start read-only access operation shall be automatically applied to that SDAI-model. When the repository containing the SDAI-model is not open, the repository shall not be opened automatically, the SDAI-model shall not be opened automatically and an error shall result;

— when an operation terminates unsuccessfully, the values of the Input parameters will be unchanged. The effect on the Output parameters is not specified in this part of ISO 10303;

— for implementations not supporting access to the SDAI data dictionary, support for the SDAI operations that specify an instance from the SDAI dictionary schema is provided by supporting SDAI language binding operations where these parameters are specified by name;

— for operations where an entity instance attribute is an input parameter, the attribute may have been declared directly in the **entity_definition** upon which the **entity_instance** is based or it may have been inherited from a supertype of that **entity_definition**. For SDAI language bindings adding support for specifying parameters by name, the case where attributes of the same name are inherited from different supertypes shall be handled by prefixing the attribute name with the name of the supertype entity type from which the attribute is inherited (see ISO 10303-11: 9.2.3.3);

— the behaviour of operations accessing an aggregate instance through its identifier or through an iterator when other operations are used to add, modify, remove or delete members of the aggregate instance is not specified in this part of ISO 10303;

— all iterators whose subject aggregate has been deleted shall be deleted at the end of an SDAI session and may be deleted earlier depending on the implementation or the dictates of the programming language of the particular binding such that the AI_NEXS or IR_NEXS error shall result when the iterator is used as a parameter to an SDAI operation;

— set, bag and list instances are managed differently from array instances by SDAI operations with respect to their bounds. The lower bound and upper bound for set, bag and list instances are considered constraints on the number of elements that a valid instance of the set, bag or list may contain. These bounds have no effect on the operations that create, add elements to or remove elements from sets, bags and lists;

— array instances are managed differently from set, bag and list instances by SDAI operations with respect to their bounds. The lower index and upper index for array instances are considered as setting both the size and valid index positions for a valid instance of the array. The size and valid index positions for an array instance are set when the array instance is created and are only changed by explicit reindex operations. As EXPRESS allows the declaration of an array index that depends on a population of the application schema, the SDAI operations allow instances of the same array type to be created that have different valid index positions. For cases where the population is not sufficient to evaluate the index expression, SDAI operations allow the index value to be specified by the application.

10.3 Environment operations

10.3.1 Open session

This operation initiates the SDAI implementation and commences a new SDAI session. The repository containing the session schema instance and data is opened by this operation and only the Close session operation can close that repository. Access to the session data is available immediately. In implementations supporting access to a data dictionary, the repositories and SDAI-models containing the data dictionary information are not opened by this operation and are not accessible until a transaction has been started.

Output

Session: **sdai_session**;
The instance of **sdai_session** created by the operation.

Possible error indicators

SS_OPN	An SDAI session is already open.
SS_NAVL	The implementation cannot open a session.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Session, a valid instance of **sdai_session** is created.

A valid instance of **implementation** is created and its identifier is set as the **Session.sdai_implementation** value.

The **Session.known_servers** attribute shall be initialized with a set instance whose members are the instances of **sdai_repository** available to the application for this session. The initialized member of **active_servers** appears in this set instance.

The **active_servers** attribute of **Session** shall be initialized with a set instance containing one member, the identifier of the repository containing the schema instance based upon the SDAI session schema and the SDAI-model containing the session data.

The **active_models** attribute of **Session** shall be initialized with a set instance containing one member, the identifier of the SDAI-model containing the session data.

The **errors** attribute of **Session** shall be initialized with a list instance containing no members. The **recording_active** attribute shall be initialized to TRUE if the implementation supports event recording, and FALSE otherwise.

The **data_dictionary** attribute of **Session** shall be initialized with the instance of **schema_instance** with which the SDAI-models comprising the SDAI data dictionary are associated if the implementation supports access to the SDAI data dictionary.

10.4 Session operations

10.4.1 Record error

This operation appends an error event to the SDAI session errors record.

Input

Session:
sdai_session;
The SDAI session in which the error takes place.

Function_id:
string_value;
A string identifying the operation with which the error is associated.

Error:
integer_value;
Error code for the error event.

Description:
string_value;
Description of the error event.

Possible error indicators

SS_NOPN	An SDAI session is not open.
ER_NSET	Event recording active is not set.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

A valid instance of **error_event** is created and appended to the list of **error_event** representing the **Session.errors** attribute.

10.4.2 Start event recording

This operation enables, or re-enables, event recording by SDAI operations in the SDAI session. Any error events previously recorded during the session remain in the error event record and any new error events

recorded are appended to the error event record.

Input

Session: **sdai_session**;
The session for which recording is to be enabled.

Output

Result: **boolean_value**;
The SDAI implementation returns TRUE if event recording is supported and enabled; FALSE if not supported.

Possible error indicators

SS_NOPN	An SDAI session is not open.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

If **Session.sdai_implementation.recording_level** = 2 specifying that event recording is supported by the SDAI implementation, **Session.recording_active** shall be set to, or remains, TRUE.

10.4.3 Stop event recording

This operation disables event recording for an SDAI session.

Input

Session: **sdai_session**;
The session for which event recording is to be disabled.

Output

Result: **boolean_value**;
TRUE if event recording is supported and disabled; FALSE if not supported.

Possible error indicators

SS_NOPN	An SDAI session is not open.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Session.recording_active shall be set to, or remains, FALSE.

10.4.4 Close session

This operation terminates the SDAI session. Further SDAI operations can be processed only after a subsequent Open session operation. In implementations supporting transaction levels 1 or 2 (see 13.1.1), the implementation shall behave as if the Close repository operation is performed on each repository in **session.active_servers**. In an implementation supporting transaction level 3, the implementation shall behave as if the End transaction access and abort operation is performed if a transaction existed in the session followed by the Close repository operation on each open repository regardless of whether a transaction existed or not.

Input

Session: **sdai_session**;
The session to be closed.

Possible error indicators

SS_NOPN	An SDAI session is not open.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

In implementations supporting transaction level 1 or 2, for each **sdai_repository** in **Session.active_servers**, function equivalent to the Close repository operation shall be applied.

In implementations supporting transaction level 3, function equivalent to the End transaction access and abort operation shall be applied for **Session.active_transaction**. For each **sdai_repository** in **Session.active_servers**, function equivalent to the Close repository operation shall be applied.

All instances of all entity types in all SDAI and application schemas shall no longer be available.

10.4.5 Open repository

This operation makes the contents of a repository available for subsequent access.

Input

Session: **sdai_session;**
The session in which repository is to be opened.

Repository: **sdai_repository**;
The repository to be opened.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RP_NEXS	The repository does not exist.
RP_NAVL	The repository is currently not available.

RP_OPN	The repository is already open.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Repository shall be added to the set in **Session.active_servers**.

10.4.6 Start transaction read-write access

This operation specifies the beginning of a sequence of operations in a session for which access is provided to **entity_instance**s such that changes can be made to those instances.

Input

Session: **sdai_session**;
The session for which read-write access is to be started.

Output

Transaction: **sdai_transaction**;
The read-write transaction that was started.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_EXS	A transaction has already been started.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Transaction, a valid instance of **sdai_transaction** is created.

Transaction.mode shall be set to **read_write**.

Transaction shall be set as **Session.active_transaction**.

10.4.7 Start transaction read-only access

This operation specifies the beginning of a sequence of operations in a session for which access is provided to **entity_instance**s such that no changes can be made to those instances.

Input

Session: **sdai_session**;
The session for which read-only access is to be started.

Output

Transaction: **sdai_transaction**;
The read-only transaction that was started.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_EXS	A transaction has already been started.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Transaction, a valid instance of **sdai_transaction** is created.

Transaction.mode shall be set to **read_only**.

Transaction shall be set as **Session.active_transaction**.

10.4.8 Commit

This operation makes persistent all changes to the contents, SDAI-models and schema instances, of all open repositories made since the last Start transaction with read-write access, Commit or Abort operation, whichever operation occurred most recently. The existing read-write transaction continues to be active. This operation performs no function in the case where the current transaction is read-only. This operation updates or sets the **change_date** attribute of any schema instance or SDAI-model that has been modified or created.

Input

Transaction: **sdai_transaction**;
The transaction that is to be committed.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NEXS	A transaction has not been started.
TR_EAB	The transaction ended abnormally.
TR_NAVL	The transaction is currently not available.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The current condition of all instances of the following in open repositories is made persistent:

— **sdai_repository_contents**;

— **schema_instance**;

— **sdai_model.name**;

— **sdai_model_contents**;

— **entity_extent**;

— **scope**;

— **application_instance**;

— **aggregate_instance** except for **non_persistent_list_instance**.

schema_instance.change_date for any modified or created **schema_instance** shall be set to the current date.

sdai_model.change_date for any modified or created **sdai_model** shall be set to the current date.

10.4.9 Abort

This operation restores the condition of the contents, SDAI-models and schema instances, of all open repositories to that which existed at the time of the last Start transaction read-write or Commit operation whichever operation occurred most recently. All deleted instances are restored, all created instances no longer exist and all modifications to instances are lost. The existing read-write transaction continues to be active. This operation performs no function in the case where the current transaction is read-only.

Input

Transaction: **sdai_transaction**;
The transaction that is to be aborted.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NEXS	A transaction has not been started.
TR_EAB	The transaction ended abnormally.
TR_NAVL	The transaction is currently not available.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The behaviour of an **iterator** on an **aggregate_instance** that was deleted, created or changed since the most recent Start transaction read-write or commit operation is not specified in this part of ISO 10303.

For instances of the following that were created since the most recent Start transaction read-write or Commit in open repositories, function equivalent to the appropriate Delete operation shall occur:

— **schema_instance**;

— **sdai_model**;

— **scope**;

— **application_instance**;

— **aggregate_instance** except for **non_persistent_list_instance**.

For instances of the following that were deleted, either directly or indirectly, since the most recent Start transaction read-write or Commit in open repositories, each instance is restored to the condition that existed at the time of the Start transaction read-write or Commit:

— **schema_instance**;

— **sdai_model**;

— **sdai_model_contents**;

— **entity_extent**;

— **scope**;

— **application_instance**;

— **aggregate_instance** except for **non_persistent_list_instance**.

The condition of all instances of the following that have been changed since the most recent Start transaction read-write or Commit in open repositories is reset to that which existed at the time of the Start transaction read-write or Commit:

— **sdai_repository_contents**;

— **schema_instance**;

— **sdai_model**;

— **sdai_model_contents**;

— **entity_extent**;

— **scope**;

— **application_instance**;

— **aggregate_instance** except for **non_persistent_list_instance**.

10.4.10 End transaction access and commit

This operation ends the sequence of operations started by the Start transaction read-write access or Start transaction read-only access operation. The implementation shall behave as if the Commit operation is performed before ending the transaction access. Further operations accessing entity instances within the session may be processed only after a subsequent Start transaction read-write access or Start transaction read-only access operation.

Input

Transaction: **sdai_transaction**;
The transaction that is to be terminated.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NEXS	A transaction has not been started.
TR_EAB	The transaction ended abnormally.
TR_NAVL	The transaction is currently not available.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Transaction shall be deleted.

10.4.11 End transaction access and abort

This operation ends the sequence of operations started by the Start transaction read-write access or Start transaction read-only access operation. The implementation shall behave as if the Abort operation is performed before ending the transaction access. Further operations accessing entity instances within the session may be processed only after a subsequent Start transaction read-write access or Start transaction read-only access operation.

Input

Transaction: **sdai_transaction**;
The transaction that is to be terminated.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NEXS	A transaction has not been started.
TR_EAB	The transaction ended abnormally.
TR_NAVL	The transaction is currently not available.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Transaction shall be deleted.

10.4.12 Create non-persistent list

This operation creates an empty **non_persistent_list_instance**. Non-persistent lists are provided as services such as a container for the results of a query and as standard mechanisms for SDAI applications to manipulate collections of entity instances. Non-persistent lists need only support lists of entity instances of entity types found in the SDAI dictionary or found in the library of functions produced in an implementation of an early binding. It need not be possible to assign the identifier of a non-persistent list to an attribute of an entity instance. All non-persistent lists shall be deleted at the end of an SDAI session and may be deleted earlier depending on the dictates of the programming language of the particular binding. The following operations are applicable to non-persistent lists:

— operations 10.12.1 to 10.12.7;

— operations 10.13.2 and 10.13.3;

— operations 10.15.1 to 10.15.3;

— operation 10.16.1;

— operations 10.19.1 to 10.19.3 and 10.19.7.

Output

List: **non_persistent_list_instance**;
The non-persistent list that was created.

Possible error indicators

SS_NOPN	An SDAI session is not open.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

List, an empty **non_persistent_list_instance**, is created.

10.4.13 Delete non-persistent list

This operation deletes a non-bounded, non-persistent list. This operation has no effect on any member of the list.

Input

List: **non_persistent_list_instance**;
The non-persistent list that was created.

Possible error indicators

SS_NOPN	An SDAI session is not open.
AI_NEXS	The non-persistent list does not exist.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

List is deleted.

10.4.14 SDAI query

This operation populates an existing non-persistent list by executing a query over a source. The source may be specified as an aggregate instance, an SDAI-model, a schema instance or a repository. In the cases where the source is an SDAI-model, schema instance or repository the source identifies a set of SDAI-models and for each SDAI-model in the set the query is executed over the **entity_extent** of the entity type defined in the logical expression of the query. After the query, the resulting aggregate will contain the instance identifiers for those entity instances where the evaluation of the logical expression defining the criteria is TRUE. If the resulting list already contained any entity instance identifiers then they remain in the list and those resulting from the query are appended. In the case that the logical expression syntax is invalid the VA_NVLD, OP_NVLD or AT_NVLD error shall result for errors in value, operator or attribute specification respectively. If the combination of attribute, value and operator is not supported the VT_NVLD error shall result.

The logical expression defining the criteria upon which inclusion in the result is determined need only be a string of the form **VALUE OPERATOR entity{.attr_spec}.attr_spec** where:

— **attr_spec** is an attribute name specification of the form: **attribute.name** or **entity_definition.name** concatenated with **attribute.name** separated by a dot (.) in the case where attributes of the same name have been inherited from multiple supertypes;

— **{.attr_spec}** is zero or more entity reference valued attribute name specifications;

— **OPERATOR** is =, <=, >=, <, >, <>, :=:, :<>:, IN, or LIKE;

— **VALUE** may be the keyword UNSET or a value as defined in table 1 where UNSET tests whether the attribute has a value or not.

This simple form may be combined with NOT, AND, OR, and parentheses to produce more complex forms. Parentheses evaluation is defined in ISO 10303-11 clause 12. The AND logical operator is defined in ISO 10303-11: 12.4.2. The OR logical operator is defined in ISO 10303-11: 12.4.3. The NOT logical operator is defined in ISO 10303-11: 12.4.1.

The value comparison operators =, <=, >=, <, > and <> are defined in ISO 10303-11: 12.2.1 with the limitation that they need not be applicable to values of aggregate or entity data types. The instance comparison operators :=: and :<>: are defined in ISO 10303-11: 12.2.2 with the limitation that they need only be applicable to values of entity data types.

The LIKE operator function, pattern matching algorithm and pattern matching characters are defined in ISO 10303-11: 12.2.5 with the exception that the left operand shall be the pattern string and the right operand shall be the target string.

The IN operator function is defined in ISO 10303-11: 12.2.3.

In the case where VALUE in the logical expression is to be a literal, the format for the literal is defined in ISO 10303-11: 7.5.

Table 1 - Attribute representations supported in query

Attribute representation	OPERATOR values	VALUE type
simple_type; **defined_type** where domain evaluates to a **simple_type**	=, <= ,>= ,<, >, <>	A literal in the corresponding domain
Nested **aggregation_type**; **defined_type** where domain evaluates to a nested **aggregation_type**	all	Not supported
entity_instance; **defined_type** where domain evaluates to an **entity_instance**	:=:, :<>:	The token ENTITY
defined_type where domain evaluates to an **enumeration_type**	=, <=, >=, <, >, <>	A string literal
string_type; **defined_type** where domain evaluates to a **string_type**	LIKE	A string literal
aggregation_type; **defined_type** where domain evaluates to an **aggregation_type**	IN	The token ENTITY or a literal of the corresponding domain

Input

Source: **query_source**;
The non-nested, persistent or non-persistent aggregate, SDAI-model, schema instance or repository.

Where: **string_value**;
The logical expression that defines the criteria upon which inclusion in the result is determined.

Entity: **entity_instance**;
The value for ENTITY in the case where the attribute being queried is a reference to an entity data type.

Result: **non_persistent_list_instance**;
The non-persistent aggregate to which the instance identifiers for those entity instances meeting the specified criteria are appended.

Output

Count: **integer_value**;
An integer count of the number of entity instances meeting the specified criteria.

Possible error indicators

SS_NOPN	An SDAI session is not open.
AI_NEXS	The input or output aggregate instance does not exist.
RP_NEXS	The repository does not exist.
MO_NEXS	The SDAI-model does not exist.
SI_NEXS	The schema instance does not exist.
EI_NEXS	The entity instance does not exist.
EI_NVLD	The logical expression entity instance is invalid.
VA_NVLD	The logical expression value is invalid.
OP_NVLD	The logical expression operator is invalid.
AT_NVLD	The logical expression attribute is invalid.
VT_NVLD	The logical expression value/operator/attribute combination is invalid.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The members of **Source** meeting the specified criteria are appended as elements of **Result**.

10.5 Repository operations

10.5.1 Create SDAI-model

This operation establishes a new **sdai_model** within which entity instances can be created and accessed. The newly created SDAI-model has no access mode associated with it.

Input

Repository: **sdai_repository**;
The repository in which the SDAI-model is to be created.

ModelName: **string_value**;
The name of the new SDAI-model.

Schema: **schema_definition**;
The schema upon which the SDAI-model shall be based.

Output

Model:	**sdai_model**; The newly created **sdai_model**.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RP_NEXS	The repository does not exist.
RP_NOPN	The repository is not open.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available
TR_EAB	The transaction ended abnormally.
MO_DUP	A duplicate SDAI-model name exists.
VT_NVLD	The SDAI-model name value type is invalid.
SD_NDEF	The schema definition is not defined.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

New instances of **sdai_model** and of **sdai_model_contents** shall be created.

Model.name shall be set to **ModelName**.

Model.underlying_schema shall be set to **Schema**.

Model.mode shall be unset.

Model.repository shall be set to **Repository**.

Model.change_date shall be set to the current date.

Model.contents shall be set to the identifier of the newly created instance of **sdai_model_contents**.

Model.contents.instances and **Model.contents.populated_folders** shall be initialized with set instances containing no members.

Model.contents.folders shall contain a single **entity_extent** instance for each **entity_definition** in **Schema.entities**, with **entity_extent.definition** set to the corresponding **entity_definition** and **entity_extent.instances** being initialized with a set instance containing no members.

Repository.contents.models shall be updated to include **Model**.

10.5.2 Create schema instance

This operation establishes a new schema instance.

Input

Repository:	**sdai_repository**; The repository in which the schema instance is to be created.
Name:	**string_value**; The name of the new schema instance.
Schema:	**schema_definition**; The schema upon which the schema instance shall be based.

Output

Instance:	**schema_instance**; The newly created schema instance.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RP_NEXS	The repository does not exist.
RP_NOPN	The repository is not open.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available
TR_EAB	The transaction ended abnormally.
SI_DUP	A duplicate schema instance name exists.
VT_NVLD	The schema instance name value type is invalid.
SD_NDEF	The schema definition is not defined.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Instance, a valid **schema_instance**, is created.

Instance.name is set to **Name**.

Instance.associated_models is initialized to a set instance with no members.

Instance.native_schema is set to **Schema**.

Instance.repository is set to **Repository**.

Instance.change_date and **Instance.validation_date** are set to the current date.

Instance.validation_result is set to FALSE.

Instance.validation_level is set to the same value as **sdai_session.sdai_implementation expression_level** for the instance of **sdai_session** where **Repository** is in **active_servers**.

Repository.contents.schemas shall be updated to include **Instance**.

10.5.3 Close repository

This operation closes an **sdai_repository** that has been previously opened. SDAI-models and schema instances within the repository are no longer available for access.

In an implementation supporting transaction levels 1 or 2 (see 13.1.1), the implementation shall behave as if an End read-only access is performed on all read-only SDAI-models. In an implementation supporting transaction level 1, the implementation shall behave as if an End read-write access on all read-write SDAI-models is performed on all active SDAI-models within the repository. In an implementation supporting transaction level 2, the implementation shall behave as if an Undo changes and End read-write access on all read-write SDAI-models is performed on all active SDAI-models within the repository.

In an implementation supporting transaction level 3, if a read-write transaction is active in the session then there are two circumstances in which this operation shall result in the TR_RW error. The first is if any SDAI-model within the repository has been created, deleted or modified since the most recent commit or abort operation was performed. The second is if any schema instance within the repository has been created, deleted or modified since the most recent commit or abort operation was performed. Otherwise, implementations supporting transaction level 3 shall behave as if the End read-only access operation is performed on all active read-only SDAI-models and the End read-write access operation is performed on all active read-write SDAI-models within the repository before the repository is closed.

Input

Repository:	**sdai_repository**; The repository to be closed.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RP_NEXS	The repository does not exist.
RP_NOPN	The repository is not open.
TR_RW	The transaction is read-write and changes are unresolved.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Repository is removed from **sdai_session.active_servers** for the current session.

All members of **Repository.contents.models** are removed from **sdai_session.active_models** for the current session.

10.6 Schema instance operations

10.6.1 Delete schema instance

This operation deletes a schema instance. If references between two SDAI-models associated with the schema instance existed and there is not another schema instance with which both SDAI-models are associated, then the references between the entity instances in those two SDAI-models are invalid (see 10.10.7).

Input

Instance: **schema_instance;**
The schema instance to be deleted.

Possible error indicators

SS_NOPN	An SDAI session is not open.
SI_NEXS	The schema instance does not exist.
RP_NOPN	The repository is not open.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Instance is deleted.

Instance shall be removed from **Instance.repository.contents.schemas**.

10.6.2 Rename schema instance

This operation assigns a new name to a schema instance.

Input

Instance: **schema_instance;**
The schema instance to rename.

Name: **string_value;**
The new name for the schema instance.

Possible error indicators

SS_NOPN	An SDAI session is not open.
SI_DUP	A duplicate schema instance name exists.
VT_NVLD	The schema instance name value type is invalid.
SI_NEXS	The schema instance does not exist.
RP_NOPN	The repository is not open.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Instance.name shall be set to **Name**.

10.6.3 Add SDAI-model

This operation adds an SDAI-model to the set of SDAI-models that are associated with a schema instance. This enables entity instances in the SDAI-model to reference and be referenced by entity instances in other SDAI-models associated with the schema instance. This also adds the entity instances in the SDAI-model to the domain for global and uniqueness rule validation defined by the schema instance. If the SDAI-model is not based upon the same schema as the schema instance but is based upon an external schema, then an entity instance in the SDAI-model shall be considered associated with the schema instance only if its entity type is defined as being domain equivalent with and entity type from the native schema upon which the schema instance is based (see A.2). If domain equivalence is not supported and the SDAI-model being added is based upon an external schema, the FN_NAVL error shall result.

Input

Instance:

schema_instance;
The schema instance with which the SDAI-model is to be associated.

Model:

sdai_model;
The SDAI-model that is to be associated with the schema instance.

Possible error indicators

SS_NOPN	An SDAI session is not open.
SI_NEXS	The schema instance does not exist.
RP_NOPN	The repository is not open.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MO_NEXS	The SDAI-model does not exist.
MO_NDEQ	The SDAI-model is not domain equivalent with the schema instance.
FN_NAVL	Domain equivalence is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Instance.associated_models shall be updated to include **Model**.

10.6.4 Remove SDAI-model

This operation removes an SDAI-model from the set of SDAI-models that are associated with a schema instance. If the SDAI-model no longer has a schema instance in common with another SDAI-model in the schema instance then all references between those two SDAI-models are invalid (see 10.10.7).

Input

Instance:

schema_instance;
The schema instance from which the SDAI-model is to be removed.

Model:

sdai_model;
The SDAI-model that is to be removed from the schema instance.

Possible error indicators

SS_NOPN	An SDAI session is not open.
SI_NEXS	The schema instance does not exist.
RP_NOPN	The repository is not open.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MO_NEXS	The SDAI-model does not exist.
MO_NVLD	The SDAI-model is associated with the schema instance.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Model shall be removed from **Instance.associated_models**.

10.6.5 Validate global rule

This operation determines whether a global rule defined in a schema is satisfied by the population associated with a schema instance. The entity instances included in the validation are all entity instances of the entity types to which the global rule applies in all of the SDAI-models that are associated with the schema instance. Entity instances within SDAI-models based upon an external schema are included in the validation if they are instances of entity types defined to be domain equivalent with entity types in the native schema by an instance of **external_schema**. Entity instances so included shall be treated as instances of the native type as defined in **domain_equivalent_type**. If the external entity type lacks properties required to satisfy the rule then the ED_NVLD error results. References to entity instances in SDAI-models that are not associated with the schema instance shall as treated if they are unset.

Input

Instance:

schema_instance;
The schema instance bounding the validation.

Rule:

global_rule;
The global rule to validate.

NonConf:

non_persistent_list_instance;
The pre-existing non-persistent list to which those instances of

where_rule within **Rule** to which **Instance** did not conform are appended if **Result** is FALSE.

Output

Result:

logical_value;
TRUE if **Rule** is satisfied, FALSE if the rule is not satisfied, and UNKNOWN if the expression evaluates to an indeterminate or UNKNOWN value.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RU_NDEF	The rule is not defined.
SI_NEXS	The schema instance does not exist.
AI_NEXS	The aggregate instance does not exist.
ED_NVLD	An external entity definition is invalid for the rule.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The rule expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.6.6 Validate uniqueness rule

This operation determines whether a uniqueness rule defined in a schema is satisfied by the population associated with a schema instance. The entity instances included in the validation are all entity instances of the entity type in which the rule was declared in all of the SDAI-models that are associated with the schema instance. Entity instances within SDAI-models based upon an external schema are included in the validation if they are instances of entity types defined to be domain equivalent with entity types in the native schema by an instance of **external_schema**. Entity instances so included shall be treated as instances of the native type as defined in **domain_equivalent_type**. If the external entity type lacks properties required to satisfy the rule then the ED_NVLD error results. References to entity instances in SDAI-models that are not associated with the schema instance shall as treated if they are unset.

Input

Instance:

schema_instance;
The schema instance bounding the validation.

Rule:

uniqueness_rule;
The uniqueness rule to be validated.

NonConf:

non_persistent_list_instance;
The pre-existing non-persistent list to which those entity instances not

conforming to the validation are appended if **Result** is FALSE.

Output

Result:

logical_value;
TRUE if the rule is satisfied, FALSE if the rule is not satisfied, and UNKNOWN if an optional explicit attribute was unset, if a derived attribute value was indeterminate or UNKNOWN or if an inverse attribute had no value.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RU_NDEF	The rule is not defined.
SI_NEXS	The schema instance does not exist.
AI_NEXS	The aggregate instance does not exist.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The attribute expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.6.7 Validate instance reference domain

This operation determines whether all attributes in the specified application instance with a reference to an entity instance as their value refer to entity instances within SDAI-models associated with the specified schema instance.

Input

Instance:

schema_instance;
The schema instance bounding the test.

Object:

application_instance;
The application instance to test.

NonConf:

non_persistent_list_instance;
The pre-existing non-persistent list to which instances of type **attribute** that reference application instances not associated with **Instance** are appended if **Result** is FALSE.

Output

Result:

logical_value;
TRUE if all the assigned attributes of **Object** are to entity instances in

Instance, FALSE if not, and UNKNOWN if any required explicit attribute values are unset that could reference an entity instance.

Possible error indicators

SS_NOPN	An SDAI session is not open.
EI_NEXS	The entity instance does not exist.
SI_NEXS	The schema instance does not exist.
AI_NEXS	The aggregate instance does not exist.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.6.8 Validate schema instance

This operation determines whether the population associated with a schema instance conforms to all constraints specified within the schema upon which the scheme instance is based. This operation updates the validation information maintained within the schema instance.

Input

Instance:

schema_instance;
The schema instance bounding the test.

Output

Result:

logical_value;
TRUE if all the constraints from the schema upon which **Instance** is based are met, FALSE if any constraint is violated, and UNKNOWN any constraint resulted in UNKNOWN.

Possible error indicators

SS_NOPN	An SDAI session is not open.
SI_NEXS	The schema instance does not exist.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
TR_NRW	The transaction is not read-write.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The **Instance.validation_date** shall be set to the current date.

The **Instance.validation_level** shall be set to the **implementation.expression_level** of the current implementation.

The **Instance.validation_result** shall be set to **Result**.

10.6.9 Is validation current

This operation determines whether complete validation of a schema instance may be required based on whether the **schema_instance.validation_result** has a value or based on whether any modification to a schema instance or any of the SDAI-models associated with the schema instance has been performed since the most recent Validate schema instance operation was performed. The validation result not being set or any modification found will result in the operation determining that validation is not current.

Input

Instance:
schema_instance;
The schema instance bounding the test.

Output

Result:
boolean_value;
TRUE if **Instance** validation result is currently set to TRUE and no modification to **Instance** or member of **Instance.associated_models** since the last validation was performed is found, otherwise FALSE.

Possible error indicators

SS_NOPN	An SDAI session is not open.
SI_NEXS	The schema instance does not exist.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.7 SDAI-model operations

10.7.1 Delete SDAI-model

This operation deletes an **sdai_model** along with all of the **entity_instances**, **aggregate_instances** and **scopes** that it contains. Any subsequent operation using a reference to the SDAI-model or to any of its contents shall behave as if the reference was unset.

Input

Model:
sdai_model;
The **sdai_model** to delete.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NEXS	A transaction has not been started.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.
MO_NEXS	The SDAI-model does not exist.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Model and **Model.contents** shall be deleted.

Each **entity_instance** in **Model.contents.instances** shall be deleted as if the Delete application instance operation had been applied (see 10.11.2).

Model shall be removed from **sdai_session.active_models** for the current session.

Model shall be removed from the **sdai_repository.contents.models** for the repository in which **Model** was created.

Model shall be removed from **schema_instance.associated_models** for all schema instances with which **Model** was associated.

Each **scope** created within **Model** shall be deleted.

10.7.2 Rename SDAI-model

This operation assigns a new name to an **sdai_model**.

Input

Model:	**sdai_model**; The **sdai_model** to rename.
ModelName:	**string_value**; The new name for the **sdai_model**.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NEXS	A transaction has not been started.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.

MO_NEXS	The SDAI-model does not exist.
VT_NVLD	The SDAI-model name value type is invalid.
MO_DUP	A duplicate SDAI-model name exists.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Model.name shall be set to **ModelName**.

10.7.3 Start read-only access

This operation makes available the instances within an SDAI-model but restricts access to them to be read-only. Any subsequent SDAI operation that attempts to modify instances within the **sdai_model** shall result in an error.

Input

Model:	**sdai_model**; The **sdai_model** to access as read-only.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NEXS	A transaction has not been started.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.
MO_NEXS	The SDAI-model does not exist.
MX_RO	The SDAI-model access is read-only.
MX_RW	The SDAI-model access is read-write.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Model.mode shall be set to **read_only**.

Model shall be added to **sdai_session.active_models** for the current session.

10.7.4 Promote SDAI-model to read-write

This operation allows read-write access to instances within an **sdai_model** to which the Start read-only access operation had been applied or that had been automatically started with read-only access as the result of a reference to an entity instance within the SDAI-model.

Input

Model:	**sdai_model**; The **sdai_model** to which read-write access is to be allowed.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NEXS	A transaction has not been started.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.
MO_NEXS	The SDAI-model does not exist.
MX_NDEF	The SDAI-model access is not defined.
MX_RW	The SDAI-model access is read-write.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Model.mode shall be set to **read_write**.

10.7.5 End read-only access

This operation ends read-only access to an **sdai_model**. Subsequent operations on the instances contained in the SDAI-model will fail until access to the SDAI-model is started by a Start read-only access or Start read-write access operation, or after automatically starting read-only access to the SDAI-model as the result of using a reference to an entity instance within the SDAI-model.

Input

Model:	**sdai_model**; The **sdai_model** to which access is to be terminated.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RP_NOPN	The repository is not open.
MO_NEXS	The SDAI-model does not exist.
MX_NDEF	The SDAI-model access is not defined.
MX_RW	The SDAI-model access is read-write.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Model.mode shall become unset.

Model shall be removed from **sdai_session.active_models** for the current session.

10.7.6 Start read-write access

This operation makes available the instances within an SDAI-model and allows access to them to be read-write.

Input

Model: **sdai_model**;
The **sdai_model** to which read-write access is to be allowed.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NEXS	A transaction has not been started.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.
MO_NEXS	The SDAI-model does not exist.
MX_RO	The SDAI-model access is read-only.
MX_RW	The SDAI-model access is read-write.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Model.mode shall be set to **read_write**.

Model shall be added to **sdai_session.active_models** for the current session.

10.7.7 End read-write access

This operation ends read-write access to an **sdai_model**. Subsequent operations on the instances contained in the SDAI-model will fail until access to the SDAI-model is started by a Start read-only access or Start read-write access operation, or after automatically starting read-only access to the SDAI-model as the result of using a reference to an entity instance within the SDAI-model. In implementations supporting transaction level 2, the implementation shall behave as if the Undo changes operation had been performed on the SDAI-model. In implementations supporting transaction level 3, if any **application_instance** or **scope** within the SDAI-model has been created, deleted or modified since the last Commit, Abort or Start transaction read-write access operation, whichever occurred most recently, this operation shall result in the TR_RW error.

Input

Model: **sdai_model**;
The **sdai_model** to which access is to be terminated.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RP_NOPN	The repository is not open.
MO_NEXS	The SDAI-model does not exist.
MX_RO	The SDAI-model access is read-only.
MX_NDEF	The SDAI-model access is not defined.

TR_RW	The transaction is read-write and changes are unresolved.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Model.mode shall be unset.

Model shall be removed from **sdai_session.active_models** for the current session.

10.7.8 Get entity definition

This operation returns the identifier of an **entity_definition** from the data dictionary based upon the entity name from the schema upon which the specified **sdai_model** is based.

Input

Model:	**sdai_model**; The **sdai_model** that is based upon the schema to be searched.
EntityName:	**string_value**; The name of the entity type of interest.

Output

Entity:	**entity_definition**; The SDAI data dictionary instance of **entity_definition** from **Model.underlying_schema.entities** that has **entity_definition.name = EntityName**.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RP_NOPN	The repository is not open.
MO_NEXS	The SDAI-model does not exist.
ED_NDEF	The entity definition name is not defined.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.7.9 Create entity instance

This operation creates a new entity instance of the specified entity data type. The attributes of the entity instance are initially unset so that the Test attribute operation returns FALSE. This operation is applicable only to instances of entity types declared in application schemas.

Input

Type:	**entity_definition**; The entity type of the instance to be created.

Model: **sdai_model**;
The **sdai_model** that will contain the entity instance.

Output

Object: **application_instance**;
The newly created entity instance.

Possible error indicators

SS_NOPN	An SDAI session is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
TR_NRW	The transaction is not read-write.
MX_NRW	The SDAI-model access is not read-write.
ED_NDEF	The entity definition is not defined.
ED_NVLD	The SDAI-model and entity definition are not based upon the same schema.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Model.contents.instances shall be updated to contain **Object**.

In the **entity_extent** in **Model.contents.folders** that has **entity_extent.definition = Type**, **entity_extent.instances** shall be updated to contain **Object**. This same **entity_extent** shall also appear in **Model.contents.populated_folders**.

10.7.10 Undo changes

This operation restores the condition of the contents, including **application_instance**s and related **scope**s, of an **sdai_model** to that which existed at the time of the last Save changes or Start read-write access operation, whichever operation occurred most recently. All created instances no longer exist, all deleted instances are restored and all modifications to instances are lost. This operation is valid only for an **sdai_model** to which read-write access is currently available. The existing read-write access to the SDAI-model continues.

Input

Model: **sdai_model**;
The **sdai_model** within which to undo changes.

Possible error indicators

SS_NOPN	An SDAI session is not open.
MO_NEXS	The SDAI-model does not exist.
MX_NRW	The SDAI-model access is not read-write.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The behaviour of an **iterator** on an **aggregate_instance** that was deleted, created or changed since the most recent Start read-write access or Undo changes operation is not specified in this part of ISO 10303.

For instances of the following that were created since the most recent Start read-write access or Save changes operation on the SDAI-model, function equivalent to the appropriate Delete operation shall occur:

— **scope**;

— **application_instance**;

— **aggregate_instance** except for **non_persistent_list_instance**.

For instances of the following that were deleted, either directly or indirectly, since the most recent Start read-write access or Save changes operation on the SDAI-model, each instance is restored to the condition that existed at the time of the Start read-write access or Save changes:

— **scope**;

— **application_instance**;

— **aggregate_instance** except for **non_persistent_list_instance**.

The condition of all instances of the following that have been changed since the most recent Start read-write access or Save changes operation on the SDAI-model is reset to that which existed at the time of the Start read-write access or Save changes operation:

— **sdai_model** except for the **sdai_model.name**;

— **sdai_model_contents**;

— **entity_extent**;

— **scope**;

— **application_instance**;

— **aggregate_instance** except for **non_persistent_list_instance**.

10.7.11 Save changes

This operation makes persistent all changes to the contents, **application_instance**s and **scope**s, of an **sdai_model** made since the last Save changes, Undo changes, or Start read-write access operation whichever operation occurred most recently. The existing read-write access continues to be active. This operation is valid only for an **sdai_model** to which read-write access is currently available. This operation updates or sets the **change_date** attribute for the specified SDAI-model.

Input

Model: **sdai_model**;
The **sdai_model** for which changes are to be saved.

Possible error indicators

SS_NOPN	An SDAI session is not open.
MO_NEXS	The SDAI-model does not exist.
MX_NRW	The SDAI-model access is not read-write.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The current condition of all instances of the following in the SDAI-model are made persistent:

— **sdai_model_contents**;

— **entity_extent**;

— **scope**;

— **application_instance**;

— **aggregate_instance** except for **non_persistent_list_instance**.

Model.change_date shall be set to the current date.

10.8 Scope operations

10.8.1 Add to scope

This operation adds an application instance to the **scope** owned by another application instance. A new **scope** is created if **Target** did not already own a **scope**. This places a restriction on the domain of valid references to **Object** to those application instances within the same **scope**.

Input

Object: **application_instance**;
The application instance to be added to a **scope**.

Target: **application_instance**;
The application instance into whose **scope Object** is to be added.

Possible error indicators

EI_NEXS	Entity instance does not exist.
TR_NRW	The transaction is not read-write.

TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

If **Target** did not already own a **scope**, a new instance of **scope** is created belonging to the SDAI-model owning **Target** and the **scope.owner** is set to **Target**. **Object** is added to the set instance representing **scope.owned**.

10.8.2 Is scope owner

This operation tests whether an application instance owns a **scope**.

Input

Object: **application_instance**;
The application instance to be tested for owning a **scope**.

Output

Result: **logical_value**;
TRUE if **Object** owns a **scope**, FALSE if **Object** does not own a **scope**, UNKNOWN if **scope** is not supported.

Possible error indicators

SS_NOPN	An SDAI session is not open.
EI_NEXS	Entity instance does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.8.3 Get scope

This operation returns the identifier of the **scope** for which the specified application instance is the owner. This operation results in the SC_NEXS error if **Object** does not own a **scope**.

Input

Object: **application_instance**;
The application instance owning a **scope**.

Output

Result: **scope**;
The **scope** for which an application instance is the owner.

Possible error indicators

SC_NEXS	Scope does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.8.4 Remove from scope

This operation removes an application instance from the specified **scope**. If the specified **scope** is nested within a higher level **scope** then the application instance is added to the next higher level **scope**. If the application instance is the last member of the set **scope.owned**, leaving the **scope** with no owned application instances, then the **scope** is deleted.

Input

Object: **application_instance**;
The application instance to be removed from Target.

Target: **scope**;
The **scope** owning Object.

Possible error indicators

EI_NEXS	The entity instance does not exist.
EI_NAVL	The entity instance is not in the scope.
SC_NEXS	Scope does not exist.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.
MX_NRW	The SDAI-model access is not read-write.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Object shall be removed from **Target.owned**; if **Object** is a member of **Target.export_list**, **Object** shall be removed from **Target.export_list**.

If **Target.owner** is owned by another **scope**, then **Object** shall be added into the owned set of that **scope**.

If **Target.owned** is empty after **Object** is removed, **Target** is deleted.

10.8.5 Add to export list

This operation extends the domain of valid references of an application instance to the next higher level. This operation adds the application instance to the export list of a **scope**. The application instance to be exported from the specified **scope** shall be a member of **Target.owned** or shall be a member of **scope.export_list** of a nested **scope** owned by an application instance that is a member of **Target.owned**.

Input

Object:	**application_instance**; The application instance to be exported.
Target:	**scope**; The **scope** into which the application instance shall be exported.

Possible error indicators

EI_NEXS	The entity instance does not exist.
EI_NAVL	The entity instance is not in the scope.
SC_NEXS	Scope does not exist.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Object shall be added to **Target.export_list**.

10.8.6 Remove from export list

This operation restores the restriction on the domain of valid references of an application instance by one level to those application instances available within the owning **scope**. This operation removes the application instance from the export list of a **scope**.

Input

Object:	**application_instance**; The application instance whose domain of valid references is to be restricted.
Target:	**scope**; The **scope** containing the application instance in its export list.

Possible error indicators

EI_NEXS	The entity instance does not exist.
EI_NAVL	The entity instance is not in the scope.
EI_NEXP	The entity instance is not exported.
SC_NEXS	Scope does not exist.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Object shall be removed from **Target.export_list**.

10.8.7 Scoped delete

This operation deletes all application instances within a **scope**. The implementation shall behave as if the Delete application instance operation had been executed on the application instance owning the specified **scope**, and on the application instances owned by that **scope**, and shall delete the specified **scope**. If any of the application instances owned by the specified **scope** are themselves **scope** owners, then these **scope**s are similarly deleted. The scoped deletion of these nested **scope**s continues until no owned application instance of any nested **scope** owns a **scope**.

Input

Object:	**scope**; The **scope** containing the application instances to be deleted.

Possible error indicators

SC_NEXS	The scope does not exist.
TR_NRW	The transaction is not read-write.
MX_NRW	The SDAI-model access is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The **Object.owner** application instance shall be deleted.

The application instances that are members of the **Object.owned** shall be deleted.

Object shall be deleted.

If any application instances that are members of the **Object.owned** set are **scope** owners then the **scope.owner**s are scope deleted as well.

10.8.8 Scoped copy

This operation creates a new **scope**, creates copies of the application instances owning and owned by the specified **scope** in the specified SDAI-model and populates the **scope.owner**, **owning** and **export_list** attributes based upon the copied application instances. **TargetModel** may be the SDAI-model within which the **scope** exists or may be another SDAI-model based upon the same schema as the one within which the **scope** exists. If any application instances owned by the specified **scope** are themselves **scope** owners, then these **scope**s are similarly copied. The scoped copy of these nested **scope**s continues until no owned application instance of any nested **scope** owns a **scope**. All references between application instances in these **scope**s are reset to reference the newly copied application instances.

Input

Object: **scope**;
The **scope** to be copied.

TargetModel: **sdai_model**;
The SDAI-model that is to contain the copied **scope** and application instances.

Output

NewObject: **scope**;
The newly created **scope**.

Possible error indicators

SC_NEXS	The scope does not exist.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
MO_NEXS	The SDAI-model does not exist.
MO_NVLD	The SDAI-model and scope are based upon different schemas.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

A new **scope** is created in **TargetModel**.

The **Object.owner** application instance is copied in **TargetModel** and set as the **NewObject.owner**.

The application instances that are members of the **Object.owned** set are copied in **TargetModel** and set as the members of the **NewObject.owned** set.

All attributes of the newly created application instances with entity instance values are reset to reference the copies of the original application instances within the new **scope**.

If any application instance that are members of the **Object.owned** set are **scope** owners then those **scope**s are scope copied in **TargetModel** as well.

10.8.9 Validate scope reference restrictions

This operation determines whether the reference restrictions of all instances in the **scope** of the identified instance are satisfied. This function validates each instance in the specified **scope** as well as in any nested **scope**.

Input

Object: **application_instance**;
The application_instance that is a **scope** owner.

Output

Result: **logical_value**;
TRUE if all restrictions are satisfied, FALSE if reference restrictions are violated; UNKNOWN if any required explicit attribute value was unset that could reference an entity instance.

Possible error indicators

EI_NEXS	The entity instance does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.9 Type operations

10.9.1 Get complex entity definition

This operation returns the constructed complex entity data type, based on the interpretation of EXPRESS AND and ANDOR (see A.1.3), that is made up of the specified entity types. The specified entity data types shall include the minimum set, containing no duplicates, of leaf entity data types required to uniquely identify the entity data type to create but may include additional supertypes of those leaf entity data types. If the SDAI implementation does not construct complex entity types when the schema is processed to populate the SDAI dictionary schema, this operation can validate and construct a new complex **entity_definition**.

Input

Types: **non_persistent_list_instance;**
The list of **entity_definition** constituent simple entity types of the complex entity data type.

Output

Complex: **entity_definition;**
The resulting complex entity definition.

Possible error indicators

SS_NOPN	An SDAI session is not open.
ED_NDEF	The entity definition is not defined.
ED_NVLD	The entity definition combination is invalid.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

If the complex entity type did not exist, **Complex**, a new instance of **entity_definition** is created.

The attribute values of **Complex** shall be set as specified in annex A.

10.9.2 Is subtype of

This operation determines whether an entity type is a subtype of another entity type. The subtype relationship shall be determined solely on the basis of information from within the data dictionary for the application schemas.

Input

Type: **entity_definition;**
The entity data type to be tested.

CompType: **entity_definition;**
The potential supertype to be tested against.

Output

Result: **boolean_value;**
The result shall be TRUE if **Type** is the same as or a subtype of **CompType**, or if there exists types A and B such that A is domain equivalent with B, **Type** is a subtype of A and B is a subtype of **CompType**; otherwise the result shall be FALSE.

Possible error indicators

SS_NOPN	An SDAI session is not open.
ED_NDEF	The entity definition is not defined.
ED_NDEQ	The entity definitions are not from domain equivalent schemas.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.9.3 Is SDAI subtype of

This operation determines whether an entity type is subtype of another entity type. The subtype relationship shall be determined on the basis of information from within the application schemas and the entity instance hierarchy defined by the SDAI parameter data schema (see clause 9).

Input

Type: **entity_definition;**
The entity data type to be tested.

CompType: **entity_definition;**
The potential supertype to be tested against.

Output

Result: **boolean_value;**
The result shall be TRUE if **Type** is the same as or a subtype of **CompType**, or if there exists types A and B such that A is domain equivalent with B and **Type** is a subtype of A and B is a subtype of **CompType**; otherwise the result shall be FALSE.

Possible error indicators

SS_NOPN	An SDAI session is not open.
ED_NDEF	The entity definition is not defined.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.9.4 Is domain equivalent with

This operation determines whether an entity data type is domain equivalent with another entity data type.

Input

Type: **entity_definition;**
The entity data type to be tested.

CompType: **entity_definition;**
The entity data type to be tested against.

Output

Result: **boolean_value;**
TRUE if **Type** is equal to or a subtype of **CompType** or if the **external_schema** instance associated with the **schema_definition** of the native schema of the **CompType** defines **Type** to be domain equivalent with **CompType**; otherwise the result shall be FALSE.

Possible error indicators

SS_NOPN	An SDAI session is not open.
ED_NDEF	The entity definition is not defined.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.10 Entity instance operations

10.10.1 Get attribute

This operation returns the value of an attribute of an **entity_instance**. If no value exists for the attribute (because it has never been assigned or has been unset, whether or not the attribute is optional) the value returned is not defined by this part of ISO 10303 and the VA_NSET error shall be generated.

If the attribute of interest is an **explicit_attribute** with an aggregate instance as its value, this operation returns the identifier of the **aggregate_instance**. If the attribute is a **derived_attribute** and the expression to derive the attribute is supported, the value is evaluated and returned according to the expression. If the evaluation result is a, possibly nested, aggregate, a, possibly nested, non-persistent list shall be created and its identifier returned. For any nested aggregate in the result, a non-persistent list shall be created. If the evaluation result is indeterminate the VA_NSET error shall be generated. If the attribute is an **inverse_attribute** and the implementation supports access, a non-persistent list containing the identifiers of the referencing entity instances is created and the identifier of the non-persistent list is returned. If there is no referencing entity instance, the non-persistent list shall be empty. If the **derived_attribute** expression is not supported or access to **inverse_attributes** is not supported, the FN_NAVL error shall be generated.

Input

Object:	**entity_instance**; The entity instance from which to obtain an attribute value.
Attribute:	**attribute**; The attribute of interest.

Output

Value:	**primitive**; The value of **Attribute** in **Object**.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EI_NEXS	The entity instance does not exist.
AT_NDEF	The attribute is not defined.
VA_NSET	The explicit value was not set or its derivation was indeterminate.

FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.10.2 Test attribute

This operation determines whether an attribute of an entity instance has a value. This operation is applicable to explicit attributes only.

NOTE - This operation is applicable whether or not that attribute is declared to be optional.

Input

Object: **entity_instance**;
The entity instance whose attribute is to be tested.

Attribute: **explicit_attribute**;
The attribute to be tested.

Output

Result: **boolean_value**;
TRUE if **Attribute** has a value in **Object**; FALSE if it does not.

Possible error indicators

SS_NOPN	An SDAI session is not open.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EI_NEXS	The entity instance does not exist.
AT_NDEF	The attribute is not defined.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.10.3 Find entity instance SDAI-model

This operation returns the identifier for the SDAI-model that contains an **entity_instance**.

Input

Object: **entity_instance**;
The instance of interest.

Output

Model: **sdai_model**;
The **sdai_model** that contains **Object**.

Possible error indicators

MO_NEXS	The SDAI-model does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EI_NEXS	The entity instance does not exist.
SY_ERR	An underlying system error occurred.

10.10.4 Get instance type

This operation returns the identifier of the **entity_definition**, as found in the SDAI data dictionary, upon which the specified **entity_instance** is based.

Input

Object: **entity_instance**;
The instance whose entity type is to be retrieved.

Output

Type: **entity_definition**;
The entity type of which **Object** is an instance.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EI_NEXS	The entity instance does not exist.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.10.5 Is instance of

This operation determines whether an **entity_instance** is an instance of exactly the specified entity data type, not one of its subtypes, or if the **entity_instance** is an instance of an entity data type defined to be domain equivalent with exactly the specified entity data type via an instance of **domain_equivalent_type** (see 6.4.8 and annex A).

Input

Object: **entity_instance**;
The entity instance to be tested.

Type: **entity_definition**;
The entity type against which to test **Object**.

Output

Result: **boolean_value**;
TRUE if the type of **Object** is the same as **Type** or if the type of **Object** is domain equivalent with **Type** as defined by an **domain_equivalent_type** of the native schema of **Type**; FALSE if it is not.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
ED_NDEF	The entity definition is not defined.
EI_NEXS	The entity instance does not exist.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.10.6 Is kind of

This operation determines whether an **entity_instance** is an instance of a particular type or of one of its subtypes including the case where it is a constituent of a complex subtype. The subtype relationship shall be determined solely on the basis of information from within the application schemas.

Input

Object: **entity_instance**;
The entity instance to be tested.

Type: **entity_definition**;
The entity type against which to test **Object**.

Output

Result: **boolean_value**;
The result shall be TRUE if **Object** is an instance of an entity type that is the same as or a subtype of **Type**, or if there exists types A and B such that A is domain equivalent with B and **Object** is a kind of A ignoring domain equivalence and B is a subtype of **Type**; otherwise the result shall be FALSE.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
ED_NDEF	The entity definition is not defined.
EI_NEXS	The entity instance does not exist.

FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.10.7 Is SDAI kind of

This operation determines whether an **entity_instance** is an instance of a particular entity type or of one of its subtypes including the case where it is a constituent of a complex subtype. The subtype relationship shall be determined on the basis of information from within the application schemas and the SDAI parameter data schema (see clause 9).

Input

Object:	**entity_instance**; The entity instance to be tested.
Type:	**entity_definition**; The entity type against which to test **Object**.

Output

Result:	**boolean_value**; The result shall be TRUE if **Object** is an instance of an entity type that is the same as or a subtype of **Type**, or if there exists types A and B such that A is domain equivalent with B and **Object** is a kind of A ignoring domain equivalence and B is a subtype of **Type**; otherwise the result shall be FALSE.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
ED_NDEF	The entity definition is not defined.
EI_NEXS	The entity instance does not exist.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.10.8 Find entity instance users

This operation returns the identifiers of all the entity instances that reference the specified entity instance within the specified set of schema instances and appends them to the resulting non-persistent list. In the case where the specified entity instance is referenced multiple times by the same referencing entity instance, the referencing entity instance shall appear in the result once for each reference.

Input

Instance:	**entity_instance**; The entity instance whose users are requested.

Domain: **non_persistent_list_instance**;
The list of the **schema_instance**s specifying the domain of entity instances to check as users of the specified Instance.

Result: **non_persistent_list_instance**;
The pre-existing non-persistent list to which the entity instances who reference **Instance** are appended.

Possible error indicators

SS_NOPN	An SDAI session is not open.
EI_NEXS	The entity instance does not exist.
SI_NEXS	The schema instance does not exist.
AI_NEXS	The list instance does not exist.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Result shall contain the **entity_instance**s that use **Instance**.

10.10.9 Find entity instance usedin

This operation returns the identifiers of all the entity instances that reference the specified entity instance in the specified role within the SDAI-models associated with the specified set of schema instances and appends them to the resulting non-persistent list. This operation is applicable to attributes whose domain is the entity type of the specified entity instance, any supertype of that entity type or any defined type that includes that entity type or any supertype of that entity type in its underlying type. In the case where the same entity instance references the specified entity instance in the specified role multiple times, the referencing entity instance shall be returned once for each reference.

Input

Instance: **entity_instance**;
The entity instance whose users are requested.

Role: **attribute**;
The attribute identifying the role being requested.

Domain: **non_persistent_list_instance**;
The list of the **schema_instance**s specifying the domain of entity instances to check as users of the specified Instance.

Result: **non_persistent_list_instance**;
The pre-existing, non-persistent list to which entity instances who reference **Instance** in the specified **Role** are appended.

Possible error indicators

SS_NOPN	An SDAI session is not open.
EI_NEXS	The entity instance does not exist.
AT_NDEF	The attribute is not defined.
SI_NEXS	The schema instance does not exist.
AI_NEXS	The list instance does not exist.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Result shall contain the **entity_instance**s that use **Instance**.

10.10.10 Get attribute value bound

This operation returns the current value of the **population_dependent_bound** for a real, string or binary type for the specified attribute value for the specified entity instance. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining the bound value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Instance:	**entity_instance**; The entity instance for which the attribute value bound is to be returned.
Attribute:	**attribute**; The **attribute** of **Instance** whose attribute value bound is to be returned.

Output

Value:	**integer_value**; The current value of the attribute value bound.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
VA_NSET	The bound value is not set.
AT_NDEF	The attribute is not defined.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The bound expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.10.11 Find instance roles

This operation returns the identifiers of all the attributes of the entity instances that reference the specified entity instance within the specified set of schema instances and appends them to the resulting non-persistent list. In the case where the specified entity instance is referenced multiple times by the same referencing attribute, the referencing attribute shall appear in the result once. In implementations supporting domain equivalence, attributes defined in external schemas may be included in the result.

NOTE - This function is similar to the EXPRESS RolesOf built-in function (see ISO 10303-11: 15.20).

Input

Instance:	**entity_instance**; The entity instance whose users are requested.
Domain:	**non_persistent_list_instance**; The list of the **schema_instance**s specifying the domain of entity instances to check as users of the specified Instance.
Result:	**non_persistent_list_instance**; The pre-existing non-persistent list to which instances of **attribute** that reference **Instance** are appended.

Possible error indicators

SS_NOPN	An SDAI session is not open.
EI_NEXS	The entity instance does not exist.
SI_NEXS	The schema instance does not exist.
AI_NEXS	The list instance does not exist.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Result shall contain the instances of **attribute** that use **Instance**.

10.10.12 Find instance data types

This operation returns the identifiers of all the **named_type**s of which the specified entity instance is a member and appends them to the resulting non-persistent list. In implementations supporting domain equivalence, the types from external schemas domain equivalent with the resulting types from the native schema for the entity instance are included in the result.

NOTE - This function is similar to the EXPRESS TypeOf built-in function (see ISO 10303-11: 15.25).

Input

Instance:	**entity_instance**; The entity instance whose types are requested.
Result:	**non_persistent_list_instance**; The pre-existing non-persistent list to which the types for **Instance** are appended.

Possible error indicators

SS_NOPN	An SDAI session is not open.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list instance does not exist.
RP_NOPN	The repository is not open.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Result shall contain the instances of **named_type** of which **Instance** is a member.

10.11 Application instance operations

10.11.1 Copy application instance

This operation creates a copy of the specified application instance in the specified SDAI-model. The values of attributes or elements of aggregates at any level of nesting in the new instance are set as follows:

— for entity instance references, the new instance will reference the same **application_instance**s as the original;

— for simple types, the values are copied;

— for aggregate instances, new aggregate instances are created.

TargetModel is the SDAI-model to contain the copy of the specified application instance. If **TargetModel** is not the SDAI-model within which **Object** exists, then it shall be based upon the same schema as the SDAI-model in which **Object** exists or it shall be based upon a schema containing an entity type defined as domain equivalent with the entity type upon which **application_instance** is based. Only attributes with the same **attribute.name** in the domain equivalent entity type are set with values. The **TargetModel** and SDAI-model in which the **Object** to be copied exists shall both be associated with the same **schema_instance** in the case where they are different.

NOTE - This operation creates references between entity instances in the two different SDAI-models if required.

Input

Object: **application_instance**;
The application instance to be copied.

TargetModel: **sdai_model**;
The SDAI-model that is to contain the copy of **Object**.

Output

NewObject: **application_instance**;
The newly created copy of **Object**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
MX_NDEF	The SDAI-model access is not defined.
MO_NDEQ	The SDAI-model is not domain equivalent with the application instance.
MO_NEXS	The SDAI-model does not exist.
EI_NEXS	The entity instance does not exist.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

For each attribute of **Object** with an aggregate instance as its value, **NewObject.values** shall contain a new aggregate instance with the same contents as that contained in **Object.values**.

For each attribute of **Object** with an entity instance as its value, **NewObject.values** shall contain the same **entity_instance** as that contained in **Object.values**.

TargetModel.contents.instances shall contain **NewObject**.

In the **entity_extent** in **TargetModel.contents.folders** that has **entity_extent.definition = Object.definition**, **entity_extent.instances** shall contain **NewObject**. This same **entity_extent** shall also appear in **TargetModel.contents.populated_folders**

10.11.2 Delete application instance

This operation deletes an application instance. All aggregate instances created as part of the application instance or as nested aggregate instances within an aggregate instance created as part of the deleted application instance are deleted. Subsequent to this operation, any attribute values of application instances defined in application or SDAI schemas that refer to the deleted application instance shall respond as if their attribute value had been unset (e.g. the Test attribute operation will return FALSE and the Remove from scope operation will result in the EI_NAVL error in cases where the deleted application instance had been in a **scope**). Any application instance referred to by the deleted application instance is not affected.

Input

Object: **application_instance**;
The application instance to be deleted.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
EI_NEXS	The entity instance does not exist.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

In the SDAI-model containing **Object**, **Object** shall be removed from **sdai_model.contents.instances**.

In the **entity_extent** in **sdai_model.contents.folders** that has **entity_extent.definition = Object.definition**, **Object** shall be removed from **entity_extent.instances**.

If **entity_extent.instances** is now empty, it shall be removed from **sdai_model.contents.populated_folders** for the SDAI-model containing **Object**.

10.11.3 Put attribute

This operation assigns a value to an explicit attribute of an application instance. In the case where the specified attribute value was previously an aggregate instance, this operation shall behave as if the Unset attribute value operation were executed on the attribute before the new attribute value is assigned.

Input

Object: **application_instance**;
The instance whose attribute is to be assigned.

Attribute: **explicit_attribute**;
The attribute to be assigned a value.

Value: **assignable_primitive**;
The new value for **Attribute** in **Object**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
EI_NEXS	The entity instance does not exist.
AT_NDEF	The attribute is not defined.

AT_NVLD	The attribute is not an explicit attribute.
VT_NVLD	The value type is invalid.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The value of **Attribute** in **Object** shall be **Value**.

10.11.4 Unset attribute value

This operation changes the state of the specified attribute, whether optional or not, so that the attribute has no value in the specified application instance. Subsequent Test attribute operations will return FALSE. If the specified attribute value was previously an aggregate instance, all aggregate instances, including nested ones, associated with the attribute in the specified application instance are deleted.

Input

Object:	**application_instance**; The instance whose attribute is to be unset.
Attribute:	**explicit_attribute**; The attribute to be unset.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
EI_NEXS	The entity instance does not exist.
AT_NDEF	The attribute is not defined.
AT_NVLD	The attribute is not an explicit attribute.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The value of the **Attribute** in **Object** shall be unset.

10.11.5 Create aggregate instance

This operation creates a new, empty aggregate instance as the representation of the specified attribute for an application instance replacing any existing attribute value. If the attribute domain is an aggregate of aggregates, only the outer-most aggregate is created. In the case where the specified attribute value was previously an aggregate instance, this operation behaves as if the Unset attribute value operation were executed on the attribute before the new aggregate is created. In the case where the domain of the specified attribute is an EXPRESS SELECT TYPE, the **aggregate_primitive** provided as input/output shall be a **select_aggregate_instance** and the **select_aggregate_instance.-data_type** shall be set on input with the **defined_type** specifying the **aggregation_type** the operation is to create. The VA_NSET error shall be set if the operation requested the creation of an array instance that is not an **application_indexed_array_-**

instance and the array instance cannot be created because the existing application schema population is not sufficient to successfully evaluate the expression defining an index value for the array instance. The EX_NSUP error shall be set if array instance is not an **application_indexed_array_instance** and the evaluation of the expression for an array index is not supported by the implementation.

Input

Object: **application_instance**;
The application instance whose attribute value is to be set.

Attribute: **explicit_attribute**;
The attribute whose value is to be set.

Input/Output

Aggregate: **aggregate_primitive**;
The aggregate instance that has been created and set as the value of **Attribute** in **Object**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
EI_NEXS	The entity instance does not exist.
AT_NDEF	The attribute is not defined.
AT_NVLD	The attribute is not an explicit attribute.
VA_NSET	The array index value is not set.
EX_NSUP	The index expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

If **Attribute** previously had an **aggregate_instance** as its value in **Object**, that **aggregate_instance**, and any nested **aggregate_instance**, shall be deleted.

The value of **Attribute** in **Object** shall be **Aggregate**, that shall be a new, empty aggregate of the appropriate type for **Attribute**.

10.11.6 Get persistent label

This operation returns a persistent label for the specified application instance. The label shall be unique within the repository containing the SDAI-model containing the application instance. Any subsequent request for a persistent label for the same application instance shall return the same persistent label in the current or any subsequent SDAI session.

Input

Object: **application_instance**;
The application instance for which a persistent label is requested.

Output

Label: **string_value**;
The persistent label for **Object**.

Possible error indicators

TR_NEXS	A transaction has not been started.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.
EI_NEXS	The entity instance does not exist.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.7 Get session identifier

This operation returns the session identifier for the application instance referenced by the specified persistent label.

Input

Label: **string_value**;
The persistent label for **Object**.

Repository: **sdai_repository**;
The repository where the application instance referenced by **Label** exists.

Output

Object: **application_instance**;
The application instance whose persistent label is **Label**.

Possible error indicators

TR_NEXS	A transaction has not been started.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.
RP_NEXS	The repository does not exist.
EI_NEXS	The entity instance does not exist.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.8 Get description

This operation returns a human readable description for the specified application instance. Any subsequent request for a description for the same application instance shall return the same description in the current SDAI session. For implementations where application instances exist in a file encoded according to ISO 10303-21, the form of this description shall be the application instance name followed by one space followed by the name of the file containing the application instance.

Input

Object:	**application_instance**; The application instance for which a description is requested.

Output

Label:	**string_value**; The description for **Object**.

Possible error indicators

TR_NEXS	A transaction has not been started.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
RP_NOPN	The repository is not open.
EI_NEXS	The entity instance does not exist.
SY_ERR	An underlying system error occurred.

10.11.9 Validate where rule

This operation determines whether a where rule is satisfied for an application instance. The rule may be one declared directly as part of the **entity_definition** upon which the application instance is based or it may be one declared on a **defined_type** that constrains the value of an attribute declared in the **entity_definition** upon which the application instance is based.

Input

Object:	**application_instance**; The instance to be validated.
Rule:	**where_rule**; The where rule to be validated for **Object**.

Output

Result:	**logical_value**; TRUE if **Rule** is satisfied, FALSE if **Rule** is violated, and UNKNOWN if **Rule** expression evaluation value is indeterminate or UNKNOWN.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
RU_NDEF	The rule is not defined.
EI_NEXS	The entity instance does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The rule expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.10 Validate required explicit attributes assigned

This operation determines whether all required explicit attributes of an application instance have values.

Input

Object: **application_instance**;
The application instance to be validated.

NonConf: **non_persistent_list_instance**;
The pre-existing non-persistent list to which the instances of **attribute** not conforming to the validation are appended if **Result** is FALSE.

Output

Result: **boolean_value**;
TRUE if all non-optional attributes of **Object** have values or if **Object** has no non-optional attributes; FALSE if any non-optional attribute has no value in **Object**.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.11 Validate inverse attributes

This operation determines whether all cardinality constraints specified in inverse attribute declarations are satisfied for an application instance.

Input

Object: **application_instance**;
The instance to be validated.

NonConf: **non_persistent_list_instance**;
The pre-existing non-persistent list to which instances of **inverse_attribute** whose members are those attributes not conforming to the validation are appended if **Result** is FALSE.

Output

Result: **boolean_value**;
TRUE if all INVERSE attribute constraints for **Object** are satisfied or if **Object** has no INVERSE attributes; FALSE if any INVERSE constraint is violated.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The bound expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.12 Validate explicit attributes references

This operation determines whether all of the entity instances that are values of attributes of an application instance are of a valid entity data type for those attributes.

Input

Object: **application_instance**;
The instance to be validated.

NonConf: **non_persistent_list_instance**;
The pre-existing non-persistent list to which instances of **attribute** whose members are those attributes not conforming to the validation are appended if **Result** is FALSE.

Output

Result: **logical_value**;
TRUE if no attribute of **Object** that has an entity instance value has a value of an incorrect type or if **Object** has no attributes that reference entity instances; FALSE if any attribute has a value that is an entity

instance of an incorrect type; and UNKNOWN if any required explicit attribute value is unset that could reference an entity instance.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.13 Validate aggregates size

This operation determines whether the number of members, or valid index values for array instances, of any attributes of the specified application instance meet the constraints specified in the declared types of these attributes. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining a bound value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Object: **application_instance**;
The instance of interest.

NonConf: **non_persistent_list_instance**;
The pre-existing non-persistent list to which instances of **attribute** whose members are those attributes not conforming to the validation are appended if **Result** is FALSE.

Output

Result: **logical_value**;
TRUE if all aggregate size constraints are satisfied for aggregate instances that are values of attributes of **Object** or if **Object** has no attribute values that are aggregate instances; FALSE if at least one size constraint is not satisfied; and UNKNOWN if a bound expression evaluation value is indeterminate or UNKNOWN.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
VA_NSET	The bound value is not set.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The bound expression evaluation is not supported by this implementation.

FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.14 Validate aggregates uniqueness

This operation determines whether all the members are unique in any aggregate instance that is a value of any attribute whose declared type requires this uniqueness. This validation is performed for all attributes of a particular instance. Uniqueness is determined by instance identifier comparison (see ISO 10303-11: 12.2.2) in cases where entity instances are the elements of the aggregate. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining a bound value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Object: **application_instance**;
The instance to be validated.

NonConf: **non_persistent_list_instance**;
The pre-existing non-persistent list to which instances of **attribute** whose members are those attributes not conforming to the validation are appended if **Result** is FALSE.

Output

Result: **boolean_value**;
TRUE if all aggregate uniqueness constraints are satisfied or **Object** had no aggregate instances as attribute values; FALSE if at least one aggregate uniqueness constraint is not satisfied.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
VA_NSET	The bound value is not set.
EX_NSUP	The bound expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.15 Validate array not optional

This operation determines whether array instances whose array type declaration does not allow optional elements have values at all index positions. This validation is performed for all attributes of the specified application instance having array instances as their values. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining a bound

value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Object: **application_instance**;
The instance to be validated.

NonConf: **non_persistent_list_instance**;
The pre-existing non-persistent list to which instances of **attribute** whose members are those attributes not conforming to the validation are appended if **Result** is FALSE.

Output

Result: **boolean_value**;
TRUE if all array instances that cannot have unset positions in fact have values at all index positions, or if all array instances representing attributes of **Object** are declared to contain optional elements, or if **Object** has no attributes with array instance values; FALSE if at least one array instance attribute value that is not declared to contain optional elements is missing at least one element or if the array instance upper or lower bound conflicts with the array type declaration.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
VA_NSET	The array index value is not set.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The index expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.16 Validate string width

This operation determines whether strings that are attribute values are of the specified width. This validation is performed for all attributes of a particular instance with a string as their value. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining the width value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Object: **application_instance**;
The instance to be validated.

NonConf: **non_persistent_list_instance**;
The pre-existing non-persistent list to which instances of **attribute** whose members are those attributes not conforming to the validation are appended if **Result** is FALSE.

Output

Result: **logical_value**;
TRUE if all attribute string values are of the correct width; FALSE if at least one attribute string value violates the declared width; and UNKNOWN if a derived attribute value expression evaluation value is indeterminate.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
VA_NSET	The bound value is not set.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The bound expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.17 Validate binary width

This operation determines whether BINARY values for attributes are of the specified width. This validation is performed for all attribute binary values of a particular instance. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining the width value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Object: **application_instance**;
The instance to be validated.

NonConf: **non_persistent_list_instance**;
The pre-existing non-persistent list to which instances of **attribute** whose members are those attributes not conforming to the validation are appended if **Result** is FALSE.

Output

Result: **logical_value**;
TRUE if all attribute binary values are of the correct width; FALSE if at least one attribute binary value violates the declared width; and

UNKNOWN if a derived attribute value expression evaluation value is indeterminate.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
VA_NSET	The bound value is not set.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The bound expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.11.18 Validate real precision

This operation determines whether real values for attributes are of the specified minimum precision. This validation is performed for all attributes of a particular instance with real values. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining the precision value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Object: **application_instance**;
The instance to be validated.

NonConf: **non_persistent_list_instance**;
The pre-existing non-persistent list to which instances of **attribute** whose members are those attributes not conforming to the validation are appended if **Result** is FALSE.

Output

Result: **logical_value**;
TRUE if all attribute real values are of at least the declared precision; FALSE if at least one attribute real value violates the declared precision; and UNKNOWN if a derived attribute value expression evaluation value is indeterminate.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
VA_NSET	The bound value is not set.
EI_NEXS	The entity instance does not exist.
AI_NEXS	The list does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The bound expression evaluation is not supported by this implementation.

FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.12 Entity instance aggregate operations

10.12.1 Get member count

This operation returns the number of elements contained in an aggregate instance, or, if the aggregate instance is an array instance, the size of the array instance.

Input

Aggregate:	**aggregate_instance**; The aggregate to be counted.

Output

Result:	**integer_value**; The number of elements currently contained in, or in the case of an array instance the size of, **Aggregate**.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

10.12.2 Is member

This operation determines whether the specified value is an element in the specified aggregate instance.

Input

Aggregate:	**aggregate_instance**; The aggregate instance against which to test.
Value:	**primitive**; The value for which to test.

Output

Result:	**boolean_value**; TRUE if **Value** is contained in **Aggregate**; otherwise FALSE.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.

AI_NEXS	The aggregate instance does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

10.12.3 Create iterator

This operation creates a new iterator over an aggregate instance. The iterator is initially set as if a Beginning operation had been performed, so that no member is referenced as the current member.

Input

Aggregate: **aggregate_instance**;
The aggregate instance for which a new iterator is to be created.

Output

Iterator: **iterator**;
The newly created iterator over **Aggregate**.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Iterator shall be a new iterator defined on **Aggregate**.

Iterator shall be positioned at the beginning of **Aggregate** and it shall have no current member.

10.12.4 Delete iterator

This operation deletes an existing iterator.

Input

Iterator: **iterator**;
The iterator to be deleted.

Possible error indicators

IR_NEXS	The iterator does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Iterator shall no longer exist.

10.12.5 Beginning

This operation positions an iterator at the beginning of its subject aggregate instance so that there is no current member. For ordered collections, this operation positions the iterator so that a Next operation will make the first member of the aggregate instance the current member. For unordered collections this operation resets the tracking by the implementation of which members have been visited by the Next operation (see 10.12.6). For unordered collections the first, or next, member determination is left to the implementation.

Input

Iterator:	**iterator**; The iterator to be positioned.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
IR_NEXS	The iterator does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Iterator shall be positioned at the beginning of **iterator.subject**, so that there is no current member.

10.12.6 Next

This operation positions an iterator so that it has a new current member. The iterator behaves as follows:

— if the iterator is at the beginning of the subject aggregate instance, this operation makes the first member the current one;

— if the iterator is positioned at the last member of the subject aggregate instance or at the end of the aggregate instance or if the aggregate instance is empty, the iterator is set to the end of the aggregate instance, and there is no current member;

— if the iterator's subject aggregate instance is an ordered collection, the new current member is the member immediately following the present one, unless the present one is the last. If this is the case, then the iterator is set so that a Previous operation will set the last member of the subject aggregate instance to be the current;

— if the iterator's subject aggregate instance is an unordered collection, the implementation tracks which members have been visited such that multiple invocations of the Next operation move the

iterator through all members of the subject aggregate instance without repetition. The determination of which of the remaining members is the next member is left to the implementation.

Input

Iterator: **iterator**;
The iterator to be positioned.

Output

Result: **boolean_value**;
TRUE if **Iterator** is set with a new current member; FALSE if **Iterator** is not set with a new current member as no succeeding aggregate member exists.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
IR_NEXS	The iterator does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Iterator shall be positioned so that its current member is the one immediately following the current member before invocation of this operation, as described above.

Iterator.position is incremented.

10.12.7 Get current member

This operation returns the value of the current member referenced by an iterator.

Input

Iterator: **iterator**;
The iterator specifying the aggregate instance and member to return.

Output

Value: **primitive**;
The current member referenced by **Iterator**.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.

AI_NSET	The aggregate instance is empty.
IR_NEXS	The iterator does not exist.
IR_NSET	The iterator has no current member set.
VA_NSET	The current member array position value is not set.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

10.12.8 Get value bound by iterator

This operation returns the value of the **population_dependent_bound** for a real, string or binary type for the aggregate element value specified by the iterator. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining the bound value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Iterator:	**iterator**; The iterator specifying the aggregate element value for which the value bound is to be returned.

Output

Value:	**integer_value**; The value of the aggregate element value bound.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
VA_NSET	The population dependent bound value is not set.
IR_NEXS	The iterator does not exist.
AI_NEXS	The aggregate instance does not exist.
VT_NVLD	The bound value type is not population dependent.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The bound expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.12.9 Get lower bound

This operation returns the value of the **population_dependent_bound** for the lower bound, or lower index, of the specified aggregate instance based upon the current population of the application schema. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining the bound value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Aggregate: **aggregate_instance**;
The aggregate instance for which the lower bound value is to be returned.

Output

Value: **integer_value**;
The value of the lower bound or lower index.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
VA_NSET	The population dependent bound value is not set.
VT_NVLD	The bound value type is not population dependent.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The bound expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.12.10 Get upper bound

This operation returns the value of the **population_dependent_bound** for the upper bound, or upper index, of the specified aggregate instance based upon the current population of the application schema. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining the bound value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Aggregate: **aggregate_instance**;
The aggregate instance for which the upper bound value is to be returned.

Output

Value: **integer_value**;
The value of the upper bound or upper index.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
VA_NSET	The population dependent bound value is not set.
VT_NVLD	The bound value type is not population dependent.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.

EX_NSUP	The bound expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.13 Application instance aggregate operations

10.13.1 Create aggregate instance as current member

This operation replaces the value of the current member of an aggregate instance with a new, empty aggregate instance. If the pre-existing value was an **aggregate_instance**, it is deleted along with any nested **aggregate_instance** it contained. The new aggregate instance replaces the pre-existing value as the current member for the specified iterator. In the case where the domain of the aggregate associated with the specified iterator is an EXPRESS SELECT TYPE, the **aggregate_primitive** provided as input/output shall be a **select_aggregate_instance** and the **select_aggregate_instance.data_type** shall be set on input to specify the TYPE the operation is to create. The VA_NSET error shall be set if the operation requested the creation of an array instance that is not an **application_indexed_array_-instance** and the array instance cannot be created because the existing application schema population is not sufficient to successfully evaluate the expression defining an index value for the array instance. The EX_NSUP error shall be set if array instance is not an **application_indexed_array_instance** and the evaluation of the expression for an array index is not supported by the implementation.

Input

Iterator:	**iterator**; The iterator on specifying the aggregate and current member to replace.

Input/Output

NewAggregate:	**aggregate_primitive** The new aggregate instance that has been created as the current member.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
IR_NEXS	The iterator does not exist.
IR_NSET	The iterator has no current member set.
VA_NSET	The population dependent array instance index value is not set.
EX_NSUP	The index expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

NewAggregate shall be added to the contents of **Iterator.subject** replacing the pre-existing **Iterator.current_member** value.

Iterator.current_member is set to reference **NewAggregate**.

If the pre-existing current member was an **aggregate_instance**, it is deleted. Any **aggregate_instance** that was a nested member of the deleted **aggregate_instance** is deleted.

10.13.2 Put current member

This operation replaces the value of the current member of an aggregate instance with the specified value. If the existing value was an **aggregate_instance**, it is deleted along with any nested **aggregate_instance** it contained. The new value replaces the pre-existing value as the current member for the specified iterator.

Input

Iterator:	**iterator**; The iterator identifying the member of the aggregate instance to be modified.
Value:	**assignable_primitive**; The value to be set in the subject member of **Iterator**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
VT_NVLD	The value type is invalid.
AI_NEXS	The aggregate instance does not exist.
AI_NSET	The aggregate instance is empty.
IR_NEXS	The iterator does not exist.
IR_NSET	The iterator has no current member set.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Value shall be added to the content of **Iterator.subject** replacing the pre-existing **Iterator.current_member** value.

Iterator.current_member is set to reference **Value**.

If the pre-existing current member was an **aggregate_instance**, it is deleted. Any **aggregate_instance** that was a nested member of the deleted **aggregate_instance** is deleted.

10.13.3 Remove current member

This operation removes the current aggregate member referenced by an iterator. The iterator is left positioned as if a Next operation had been invoked before the member was removed. This operation shall fail when applied to an **array_instance**. If referenced member was an **aggregate_instance**, it is deleted along with any nested **aggregate_instance** it contained.

Input

Iterator: **iterator**;
The iterator identifying the aggregate and member to be removed.

Output

Result: **boolean_value**;
TRUE if **Iterator** is set with a new current member; FALSE if **Iterator** is not set with a new current member as no succeeding aggregate member exists.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is an array.
IR_NEXS	The iterator does not exist.
IR_NSET	The iterator has no current member set.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Iterator.current_member shall be removed from **Iterator.subject**.

Iterator.current_member shall be positioned as if a Next operation had been applied before the pre-existing member was removed.

10.14 Application instance unordered collection operations

10.14.1 Add unordered

This operation adds a value as a member of an unordered collection.

Input

Aggregate: **unordered_collection**;
The bag or set into which **Value** shall be added.

Value: **assignable_primitive**;
The value to be added to **Aggregate**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.

TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an unordered collection.
VT_NVLD	The value type is invalid.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Value shall be added as a member of **Aggregate.contents**.

10.14.2 Create aggregate instance unordered

This operation adds a new, empty aggregate instance as a member of an existing unordered collection. In the case where the domain of the specified aggregate instance is an EXPRESS SELECT TYPE, the **aggregate_primitive** provided as input/output shall be a **select_aggregate_instance** and the **select_aggregate_instance.data_type** shall be set on input to specify the TYPE the operation is to create. The VA_NSET error shall be set if the operation requested the creation of an array instance that is not an **application_indexed_array_instance** and the array instance cannot be created because the existing application schema population is not sufficient to successfully evaluate the expression defining an index value for the array instance. The EX_NSUP error shall be set if array instance is not an **application_indexed_array_instance** and the evaluation of the expression for an array index is not supported by the implementation.

Input

Aggregate: **unordered_collection**;
The aggregate instance into which a value is to be added.

Input/Output

NewAggregate: **aggregate_primitive**;
The new aggregate instance that has been added to **Aggregate**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an unordered collection.
VA_NSET	The population dependent array instance index value is not set.
EX_NSUP	The index expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

NewAggregate shall be a new, empty aggregate instance of the member type of **Aggregate**.

NewAggregate shall be added as a member of **Aggregate.contents**.

10.14.3 Remove unordered

This operation removes one occurrence of the specified value from the contents of an unordered collection. If the member being removed is an aggregate instance, it and any nested aggregate instances are deleted.

Input

Aggregate:	**unordered_collection**; The bag or set from which **Value** shall be removed.
Value:	**primitive**; The value to be removed from **Aggregate**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an unordered collection.
VA_NEXS	The value does not exist in the aggregate instance.
VT_NVLD	The value type is invalid.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Value shall be removed as a member of **Aggregate.contents**.

10.15 Entity instance ordered collection operations

10.15.1 Get by index

This operation returns the value of the member at a particular index position in an ordered collection.

Input

Aggregate:	**ordered_collection**; The array or list from which to return a value.
Index:	**integer_value**; The array index or list position from which the value is to be returned.

Output

Value: **primitive**;
The value at the position **Index** in **Aggregate**.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an ordered collection.
IX_NVLD	The index position is invalid.
VT_NVLD	The value type is not integer.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

10.15.2 End

This operation positions an iterator on an ordered collection so that there is no current member and so that a Previous operation will cause the last member in the ordered collection to become the current member.

Input

Iterator: **iterator**;
The iterator to be positioned.

Possible error indicators

AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an ordered collection.
IR_NEXS	The iterator does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Iterator shall be positioned at the end of **Iterator.subject**, such that there is no current member.

10.15.3 Previous

This operation sets the preceding member of an ordered collection as the new current member. If the iterator was at the end of the subject aggregate instance in which case there is no current member, this operation causes the last member of the aggregate instance to become the current member. If the iterator was positioned at the first member of the subject aggregate instance, or at the beginning of the aggregate instance or if the aggregate instance is empty, the iterator is set to the beginning of the aggregate instance, and there is no current member.

Input

Iterator: **iterator**;
The iterator to be positioned.

Output

Result: **boolean_value**;
TRUE if there is a member at the new current position; it returns FALSE if the iterator has been positioned at the beginning of the aggregate.

Possible error indicators

AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an ordered collection.
IR_NEXS	The iterator does not exist.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Iterator shall be positioned such that its current member is the one immediately preceding the current member before invocation of this operation, as described above.

10.15.4 Get value bound by index

This operation returns the value of the **population_dependent_bound** for a real, string or binary type for the aggregate element value at the specified index position. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining the bound value. The EX_NSUP error shall be set if the evaluation of the expression is not supported by the implementation.

Input

Aggregate: **ordered_collection**;
The aggregate instance containing the element value for which the value bound is to be returned.

Index: **integer_value**;
The index position specifying the aggregate element value for which the value bound is to be returned.

Output

Value: **integer_value**;
The value of the aggregate element value bound.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
VA_NSET	The population dependent aggregate bound value is not set.
IX_NVLD	The index position is invalid.
VT_NVLD	The value type is invalid.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The bound expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.16 Application instance ordered collection operations

10.16.1 Put by index

This operation replaces a member value at a particular index position in an ordered collection. If the previous value was an **aggregate_instance**, it is deleted along with any nested **aggregate_instance** it contained.

Input

Aggregate:	**ordered_collection**; The array or list to be modified.
Index:	**integer_value**; The array index or list position for the member.
Value:	**assignable_primitive**; The value to be set into **Aggregate** at **Index**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an ordered collection.
IX_NVLD	The index position is invalid.
VT_NVLD	The value type is invalid.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The member of **Aggregate** at position **Index** shall be **Value**.

10.16.2 Create aggregate instance by index

This operation replaces an existing member of an ordered collection with a new, empty aggregate instance. The member to replace is specified by its integer index position within the ordered collection. If the pre-existing member was an aggregate instance, it is deleted along with any nested **aggregate_instance** it contained. In the case where the domain of the specified aggregate instance is an EXPRESS SELECT TYPE, the **aggregate_primitive** provided as input/output shall be a **select_aggregate_instance** and the **select_aggregate_instance.data_type** shall be set on input to specify the TYPE the operation is to create. The VA_NSET error shall be set if the operation requested the creation of an array instance that is not an **application_indexed_array_instance** and the array instance cannot be created because the existing application schema population is not sufficient to successfully evaluate the expression defining an index value for the array instance. The EX_NSUP error shall be set if array instance is not an **application_indexed_array_instance** and the evaluation of the expression for an array index is not supported by the implementation.

Input

Aggregate: **ordered_collection**;
The aggregate instance containing the member to be replaced.

Index: **integer_value**;
The array index or list position for the member to be replaced.

Input/Output

NewAggregate: **aggregate_primitive**;
The new aggregate instance that has been added to **Aggregate** at the position specified by **Index**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an ordered collection.
IX_NVLD	The index position is invalid.
VA_NSET	The population dependent array instance index value is not set.
EX_NSUP	The index expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

NewAggregate shall be a new, empty aggregate instance of the member type of **Aggregate**.

NewAggregate shall be a member of **Aggregate**, at the position specified by **Index**.

10.17 Entity instance array operations

10.17.1 Test by index

This operation determines whether a value is set for the member at a particular index position in an array instance.

Input

Aggregate: **array_instance**;
The array instance to be tested.

Index: **integer_value**;
The index position to be tested in **Aggregate**.

Output

Result: **boolean_value**;
TRUE if a value is set at the position **Index** in **Aggregate**; FALSE if no value is set.

Possible error indicators

AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an array.
IX_NVLD	The index position is invalid.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

10.17.2 Test current member

This operation determines whether a value is set for the member at a particular position in an array instance as specified by an iterator.

Input

Iterator: **iterator**;
The iterator specifying the position in the array to be tested.

Output

Result: **boolean_value**;
TRUE if a value is set at the position specified by **Iterator** in the array; FALSE if no value is set.

Possible error indicators

AI_NEXS	The aggregate instance does not exist.

AI_NVLD	The aggregate instance is not an array.
IR_NEXS	The iterator does not exist.
IR_NSET	The iterator has no current member set.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
SY_ERR	An underlying system error occurred.

10.17.3 Get lower index

This operation returns the value of the **population_dependent_bound** for the lower index of the specified array instance based upon the population of the application schema that existed when the array instance was created, or as modified by the most recent Reindex array or Reset array index operations on the specified array instance.

Input

Array: **array_instance**;
The array instance for which the lower index value is to be returned.

Output

Value: **integer_value**;
The value of the lower index.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not a population dependent array.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.17.4 Get upper index

This operation returns the value of the **population_dependent_bound** for the upper index of the specified array instance based upon the population of the application schema that existed when the array instance was created, or as modified by the most recent Reindex array or Reset array index operations on the specified array instance.

Input

Array: **array_instance**;
The array instance for which the upper index value is to be returned.

Output

Value:	**integer_value**; The value of the upper index.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not a population dependent array.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.18 Application instance array operations

10.18.1 Unset value by index

This operation changes the value of a member of the specified array instance at the specified position so that the array instance has no value at that position. The position is specified by an index. Subsequent Test Value operations at this position will return FALSE. If the previous value was an **aggregate_instance**, it is deleted along with any nested **aggregate_instance** it contained.

Input

Aggregate:	**array_instance**; The array to be modified.
Index:	**integer_value**; The array index of the member to be unset.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an array.
IX_NVLD	The index position is invalid.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The member of **Aggregate** at **Index** shall be unset.

10.18.2 Unset value current member

This operation changes the value of a member of the specified array instance at the specified position so that the array instance has no value at that position. The position is specified by an iterator. Subsequent Test Value operations at this position will return FALSE. If the previous value was an **aggregate_instance**, it is deleted along with any nested **aggregate_instance** it contained. This operation can be applied only to an iterator whose subject is an array instance.

Input

Iterator:	**iterator**; The iterator specifying the array instance and position to be unset.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not an array.
IR_NEXS	The iterator does not exist.
IR_NSET	The iterator has no current member set.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The current member of **Iterator.subject** referenced by **Iterator** shall be unset.

10.18.3 Reindex array

This operation resizes the specified array instance based upon the **population_dependent_bound** value for the lower or upper index, or both, based upon the current population of the application schema. After the operation has completed successfully, the specified array instance size is based upon the new index values and only array positions corresponding to the new index values are valid. Any array elements at an index position that remains valid are not affected by this operation. Any array elements at an index position that is no longer valid are no longer accessible. If an array element value was an aggregate instance, then that aggregate instance along with any other nested aggregate instances that it contained are deleted. Any new array element index positions resulting from this operation have an unset value. The VA_NSET error shall be set if the existing application schema population is not sufficient to successfully evaluate the expression defining an index value.

Input

Array:	**array_instance**; The array instance to be reindexed.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not a population dependent array.
VA_NSET	The population dependent array instance index value is not set.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
EX_NSUP	The index expression evaluation is not supported by this implementation.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.18.4 Reset array index

This operation resets the value of the upper and lower index for the specified array instance. The array instance shall be one whose index values are based upon a **population_dependent_bound**. After the operation has completed successfully, the specified array instance size is based upon the new index values and only array positions corresponding to the new index values are valid. Any array elements at an index position that remains valid are not affected by this operation. Any array elements at an index position that is no longer valid are no longer accessible. If an inaccessible array element value was an aggregate instance, then that aggregate instance along with any other nested aggregate instances that it contained is deleted. Any new array element index positions resulting from this operation have an unset value. The upper index value shall be greater than or equal to the lower index value.

Input

Array:	**array_instance**; The array instance to be reindexed.
Lower:	**integer_value**; The value for the new lower index.
Upper:	**integer_value**; The value for the new upper index.

Possible error indicators

MX_NDEF	The SDAI-model access is not defined.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not a population dependent array.
VA_NVLD	The upper index is less than the lower index.
VT_NVLD	The value type is not integer.
TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
FN_NAVL	This function is not supported by this implementation.
SY_ERR	An underlying system error occurred.

10.19 Application instance list operations

10.19.1 Add before current member

This operation adds a member to a list instance referenced by an iterator. The new member is added immediately before the current member. The current member position is unchanged. If the iterator is at the beginning or end, the new member is made the first or last member, respectively. If the list instance is empty, the new member is added and the iterator is set as if the End operation had been performed.

Input

Iterator:	**iterator**; The iterator specifying the list instance and position for insertion.
Value:	**assignable_primitive**; The value to be inserted into **Iterator.subject** immediately before the current member.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
VT_NVLD	The value type is invalid.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not a list.
IR_NEXS	The iterator does not exist.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The member of **Iterator.subject** immediately before the current member referenced by **Iterator** shall be **Value**.

No member shall be removed from **Iterator.subject** by this operation.

10.19.2 Add after current member

This operation adds a member to a list instance referenced by an iterator. The new member is added immediately after the current member. The current member position is unchanged. If the iterator is at the beginning or end, the new member is made the first or last member, respectively. If the list instance is empty, the new member is added and the iterator is set as if the Beginning operation had been performed.

Input

Iterator:	**iterator**; The iterator specifying the list instance and position for insertion.

Value: **assignable_primitive**;
The value to be inserted into **Iterator.subject** immediately after the current member.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
VT_NVLD	The value type is invalid.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not a list.
IR_NEXS	The iterator does not exist.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The member of **Iterator.subject** immediately after the current member referenced by **Iterator** shall be **Value**.

No member shall be removed from **Iterator.subject** by this operation.

10.19.3 Add by index

This operation adds a new member to a list instance. The position of the new member within the list instance is specified by the given index. In the case where the value of the specified index is equal to the count of members in the specified list instance plus one, the specified value is appended to the list instance. Any specified index value greater than the count of members in the specified list instance plus one is invalid.

Input

Aggregate: **list_instance**;
The list instance to be modified.

Index: **integer_value**;
The list position for the new member.

Value: **assignable_primitive**;
The value to be added into **Aggregate**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.

AI_NVLD	The aggregate instance is not a list.
IX_NVLD	The index position is invalid.
VT_NVLD	The value type is invalid.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

Value shall be added to **Aggregate** at position **Index**.

10.19.4 Create aggregate instance before current member

This operation adds a new, empty aggregate instance as a member of an existing aggregate instance. The new aggregate is added immediately before the position indicated by the iterator. If the iterator is at the beginning or end, the new aggregate instance is made the first or last member, respectively. The current member of the iterator is unchanged. In the case where the domain of the aggregate instance associated with the specified iterator is an EXPRESS SELECT TYPE, the **aggregate_primitive** provided as input/output shall be a **select_aggregate_instance** and the **select_aggregate_instance.-data_type** shall be set on input to specify the TYPE the operation is to create. The VA_NSET error shall be set if the operation requested the creation of an array instance that is not an **application_indexed_array_instance** and the array instance cannot be
created because the existing application schema population is not sufficient to successfully evaluate the expression defining an index value for the array instance. The EX_NSUP error shall be set if array instance is not an **application_indexed_array_instance** and the evaluation of the expression for an array index is not supported by the implementation.

Input

Iterator: **iterator**;
The iterator on **Aggregate** specifying the position before which a value is to be added.

Input/Output

NewAggregate: **aggregate_primitive**
The new aggregate instance that has been added to **Aggregate** before the position specified by **Iterator**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not a list.
IR_NEXS	The iterator does not exist.
IR_NSET	The iterator has no current member set.
VA_NSET	The population dependent array instance index value is not set.

EX_NSUP	The index expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

NewAggregate shall be a new, empty aggregate instance of the member type of **Aggregate**.

NewAggregate shall be a member of **Aggregate**, before the position specified by **Iterator**.

10.19.5 Create aggregate instance after current member

This operation adds a new, empty aggregate instance as a member of an existing aggregate instance. The new aggregate instance is added immediately after the position indicated by the iterator. If the iterator is at the beginning or end, the new aggregate is made the first or last member, respectively. The current member of the iterator is unchanged. In the case where the domain of the aggregate instance associated with the specified iterator is an EXPRESS SELECT TYPE, the **aggregate_primitive** provided as input/output shall be a **select_aggregate_instance** and the **select_aggregate_instance.data_type** shall be set on input to specify the TYPE the operation is to create. The VA_NSET error shall be set if the operation requested the creation of an array instance that is not an **application_indexed_array_instance** and the array instance cannot be created because the existing application schema population is not sufficient to successfully evaluate the expression defining an index value for the array instance. The EX_NSUP error shall be set if array instance is not an **application_indexed_array_instance** and the evaluation of the expression for an array index is not supported by the implementation.

Input

Iterator:	**iterator**; The iterator on **Aggregate** specifying the position after which a value is to be added.

Input/Output

NewAggregate:	**aggregate_primitive** The new aggregate instance that has been added to **Aggregate** after the position specified by **Iterator**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The aggregate instance does not exist.
AI_NVLD	The aggregate instance is not a list.
IR_NEXS	The iterator does not exist.
IR_NSET	The iterator has no current member set.
VA_NSET	The population dependent array instance index value is not set.
EX_NSUP	The index expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

NewAggregate shall be a new, empty aggregate instance of the member type of **Aggregate**.

NewAggregate shall be a member of **Aggregate**, after the position specified by **Iterator**.

10.19.6 Add aggregate instance by index

This operation adds a new, empty aggregate instance as a member of an existing aggregate instance. The new member's position within the list instance is specified by the specified index. In the case where the value of the specified index is equal to the count of members in the specified aggregate instance plus one, the new aggregate instance is appended to the list instance. In the case where the domain of the specified aggregate instance is an EXPRESS SELECT TYPE, the **aggregate_primitive** provided as input/output shall be a **select_aggregate_instance** and the **select_aggregate_instance.-data_type** shall be set on input to specify the TYPE the operation is to create. The VA_NSET error shall be set if the operation requested the creation of an array instance that is not an **application_indexed_array_instance** and the array instance cannot be created because the existing application schema population is not sufficient to successfully evaluate the expression defining an index value for the array instance. The EX_NSUP error shall be set if array instance is not an **application_indexed_array_instance** and the evaluation of the expression for an array index is not supported by the implementation.

Input

Aggregate:	**list_instance**; The list instance to be modified.
Index:	**integer_value**; The list position for the new member.

Input/Output

NewAggregate:	**aggregate_primitive** The new aggregate instance that has been added to **Aggregate** before the position specified by **Iterator**.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The list instance does not exist.
AI_NVLD	The aggregate instance is not a list.
IX_NVLD	The index position is invalid.
VA_NSET	The population dependent array instance index value is not set.
EX_NSUP	The index expression evaluation is not supported by this implementation.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

NewAggregate shall be a new, empty aggregate instance of the member type of **Aggregate** at position **Index**.

10.19.7 Remove by index

This operation removes the list member at the position specified by the given index. If the list member is an aggregate instance, that aggregate instance is deleted along with all aggregate instances nested within the deleted aggregate instance.

Input

Aggregate:	**list_instance**; The list instance to be modified.
Index:	**integer_value**; The list position of the member to be removed.

Possible error indicators

TR_NRW	The transaction is not read-write.
TR_NAVL	The transaction is currently not available.
TR_EAB	The transaction ended abnormally.
MX_NRW	The SDAI-model access is not read-write.
AI_NEXS	The list instance does not exist.
AI_NVLD	The aggregate instance is not a list.
IX_NVLD	The index position is invalid.
SY_ERR	An underlying system error occurred.

Effect on the SDAI environment

The member of **Aggregate** at **Index** shall be removed.

If the member removed **Aggregate** is itself an **aggregate_instance**, it along with all its nested **aggregate_instance**s is deleted.

11 SDAI errors

Table 2 specifies a set of standard error indicator constants along with a description of the error and the error code value that shall be set as the **error_event.error** value when the error occurs. The error indicator constant name is formed by combining a target with a predicate separated by an underscore. For example: MO_NOPN for SDAI-model not open. For the purposes of table 2 and the possible error indicators specified for each operation in clause 10, the following target abbreviations apply.

AI	aggregate instance
AT	attribute

ED	entity definition
EI	entity instance
ER	event recording
EX	expression
FN	function
IR	iterator
IX	index
OP	operator
RP	repository
RU	rule
SC	scope
SD	schema definition
SI	schema instance
SS	session
MO	SDAI-model
MX	SDAI-model access
TR	transaction
SY	underlying system
VA	value
VT	value type

For the purposes of table 2 and the possible error indicators specified for each operation in clause 10, the following predicate abbreviations apply.

DUP	duplicate
EAB	ended abnormally
ERR	error
EXS	exists
NAVL	not available
NDEF	not defined
NDEQ	not domain equivalent
NEXP	not exported
NEXS	not exist
NOPN	not open
NRW	not read-write
NSET	not set or empty
NSUP	not supported
NVLD	invalid
OPN	is open
RO	read-only
RW	read-write

The enumeration of the set of error indicators that an SDAI implementation shall support is defined in table 2.

Table 2 - SDAI error indicators

Error indicator	Description	Error code	Error base
SS_OPN	Session open	10	sdai_session
SS_NAVL	Session not available	20	
SS_NOPN	Session is not open	30	
RP_NEXS	Repository does not exist	40	
RP_NAVL	Repository not available	50	sdai_repository
RP_OPN	Repository open	60	sdai_repository
RP_NOPN	Repository is not open	70	sdai_repository
TR_EAB	Transaction ended abnormally	80	sdai_transaction
TR_EXS	Transaction exists	90	sdai_transaction
TR_NAVL	Transaction currently not available	100	sdai_transaction
TR_RW	Transaction read-write	110	sdai_transaction
TR_NRW	Transaction not read-write	120	sdai_transaction
TR_NEXS	Transaction does not exist	130	
MO_NDEQ	SDAI-model not domain equivalent	140	sdai_model
MO_NEXS	SDAI-model does not exist	150	
MO_NVLD	SDAI-model invalid	160	sdai_model
MO_DUP	SDAI-model duplicate	170	sdai_model
MX_NRW	SDAI-model access not read-write	180	sdai_model
MX_NDEF	SDAI-model access not defined	190	sdai_model
MX_RW	SDAI-model access read-write	200	sdai_model
MX_RO	SDAI-model access read-only	210	sdai_model
SD_NDEF	Schema definition not defined	220	

Table 2 - (continued)

Error indicator	Description	Error code	Error base
ED_NDEF	Entity definition not defined	230	
ED_NDEQ	Entity definition not domain equivalent	240	
ED_NVLD	Entity definition invalid	250	entity_definition
RU_NDEF	Rule not defined.	260	
EX_NSUP	Expression evaluation not supported	270	bound
AT_NVLD	Attribute invalid	280	attribute
AT_NDEF	Attribute not defined	290	
SI_DUP	Schema instance duplicate	300	schema_instance
SI_NEXS	Schema instance does not exist	310	
EI_NEXS	Entity instance does not exist	320	
EI_NAVL	Entity instance not available	330	application_instance
EI_NVLD	Entity instance invalid	340	entity_instance
EI_NEXP	Entity instance not exported	350	application_instance
SC_NEXS	Scope does not exist	360	
SC_EXS	Scope exists	370	
AI_NEXS	Aggregate instance does not exist	380	
AI_NVLD	Aggregate instance invalid	390	aggregate_instance
AI_NSET	Aggregate instance is empty	400	aggregate_instance
VA_NVLD	Value invalid	410	
VA_NEXS	Value does not exist	420	
VA_NSET	Value not set	430	
VT_NVLD	Value type invalid	440	

Table 2 - (concluded)

Error indicator	Description	Error code	Error base
IR_NEXS	Iterator does not exist	450	
IR_NSET	Current member is not defined	460	iterator
IX_NVLD	Index invalid	470	aggregate_instance
ER_NSET	Event recording not set	480	
OP_NVLD	Operator invalid	490	
FN_NAVL	Function not available	500	
SY_ERR	Underlying system error	1000	

12 SDAI state model

The SDAI State Models describe several of the transitions from one state to another that occur as the result of the successful execution of an SDAI operation. A state, for the purposes of this clause, is described by the SDAI operations that are valid at a specific point in time in an SDAI session. This describes the sequence of operations that is required within an SDAI session to perform a specific task. The successful execution of one SDAI operation may or may not change the set of valid, available operations. For example, before the Open Session operation is executed the only valid, available SDAI operation is Open Session. Once the Open Session operation has been successfully executed, then the Close Session operation is valid and available. Therefore, Open Session has changed the state of the SDAI session.

There are three SDAI State Models described in this clause. Each is based upon a particular level of support for transactions as described in 13.1.1. Each state in each SDAI State Model has a specific set of operations that are available. Table 3 specifies a grouping of the SDAI operations into categories. These categories are then used to describe the operations that are available in each state.

NOTE - The states that depend on the existence of entity instances, aggregate instances, scopes and iterators are not defined in this clause. The availability of these operations depends on the existence of the required instance input parameters and the read-only or read-write access available to the specific instance.

Table 3 - SDAI operations groupings

Category	Description	Operations
A	Initiate	10.3.1 Open session
B	Level 3 Start	10.4.6 Start transaction read-write access 10.4.7 Start transaction read-only access
C	Session	10.4.1 Record error 10.4.2 Start event recording 10.4.3 Stop event recording 10.4.4 Close session 10.4.12 Create non-persistent list 10.4.13 Delete non-persistent list 10.4.14 SDAI query
D	Open Repository	10.4.5 Open repository
E	Level 3 Persistence	10.4.8 Commit 10.4.9 Abort 10.4.10 End transaction access and commit 10.4.11 End transaction access and abort
F	Repository Close	10.5.3 Close repository
G	Model Read	10.7.3 Start read-only access
H	Schema Instance Read	10.6.5 Validate global rule 10.6.6 Validate uniqueness rule 10.6.7 Validate instance reference domain 10.6.9 Is validation current
I	Create objects	10.5.1 Create SDAI-model 10.5.2 Create schema instance
J	Model Write	10.7.6 Start read-write 10.7.1 Delete SDAI-model 10.7.2 Rename SDAI-model
K	Schema Instance Write	10.6.3 Add SDAI-model 10.6.4 Remove SDAI-model 10.6.1 Delete schema instance 10.6.2 Rename schema instance 10.6.8 Validate schema instance

Table 3 - (continued)

Category	Description	Operations
L	Dictionary Model	10.9.1 Get Complex Entity. 10.9.2 Is subtype of 10.9.3 Is SDAI subtype of 10.9.4 Is domain equivalent with 10.9.5 Get population dependent bound
M	Entity Instance Read	10.7.8 Get entity definition 10.10.1 Get attribute 10.10.2 Test attribute 10.10.3 Find entity instance SDAI-model 10.10.4 Get instance type 10.10.5 Is instance of 10.10.6 Is kind of 10.10.7 Is SDAI kind of 10.10.8 Find entity instance users 10.10.9 Find entity instance usedin 10.10.10 Get attribute value bound 10.10.11 Find instance roles 10.10.12 Find instance data types 10.11.6 Get persistent label 10.11.7 Get session identifier 10.11.8 Get description 10.11.9 Validate where rule 10.11.10 Validate required explicit attributes assigned 10.11.11 Validate inverse attributes 10.11.12 Validate explicit attributes references 10.11.16 Validate string width 10.11.17 Validate binary width 10.11.18 Validate real precision

Table 3 - (continued)

Category	Description	Operations
N	Aggregate Read	10.11.13 Validate aggregates size 10.11.14 Validate aggregates uniqueness 10.11.15 Validate array not sparse 10.12.1 Get member count 10.12.2 Is member 10.12.3 Create iterator 10.12.9 Get lower bound 10.12.10 Get upper bound 10.15.1 Get by index 10.15.4 Get value bound by index 10.17.1 Test by index 10.17.3 Get lower index 10.17.4 Get upper index
O	Iterator Read	10.12.4 Delete iterator 10.12.5 Beginning 10.12.6 Next 10.12.7 Get current member 10.12.8 Get value bound by iterator 10.15.2 End 10.15.3 Previous 10.17.2 Test current member
P	Scope Read	10.8.2 Is scope owner 10.8.3 Get scope 10.8.9 Validate scope reference restrictions
Q	End Model RO	10.7.5 End read-only access
R	Promote Model	10.7.4 Promote SDAI-model to read-write
S	Level 2 Persistence	10.7.10 Undo changes 10.7.11 Save changes
T	Model Write	10.7.7 End read-write 10.7.9 Create entity instance

Table 3 - (concluded)

Category	Description	Operations
U	Application Instance Write	10.11.1 Copy application instance 10.11.2 Delete application instance 10.11.3 Put attribute. 10.11.4 Unset attribute value 10.11.5 Create aggregate instance
V	Aggregate Instance Write	10.14.1 Add unordered 10.14.2 Create aggregate instance unordered 10.14.3 Remove unordered 10.16.1 Put by index 10.16.2 Create aggregate instance by index 10.18.1 Unset value by index 10.18.3 Reindex array 10.18.4 Reset array index 10.19.3 Add by index 10.19.6 Add aggregate instance by index 10.19.7 Remove by index
W	Iterator Write	10.13.1 Create aggregate instance as current member 10.13.2 Put current member 10.13.3 Remove current member 10.18.2 Unset value current member 10.19.1 Add before current member 10.19.2 Add after current member 10.19.4 Create aggregate instance before current member 10.19.5 Create aggregate instance after current member
X	Scope Write	10.8.1 Add to scope 10.8.4 Remove from scope 10.8.5 Add to export list 10.8.6 Remove from export list 10.8.7 Scoped delete 10.8.8 Scoped copy

12.1 State model for transaction level 1

The state model for transaction level 1 implementations consists of five states. These states along with the possible transitions between these states are specified in 12.1.1 through 12.1.6. Operations in categories B, E and S need not be supported in transaction level 1.

12.1.1 No Session 1 State

The No Session 1 State occurs before the Open Session operation has been invoked. Operations in category A are available in this state. Operations in categories C, D, F, G, H, I, J, K, L, M, N, O, P, Q, R, T, U, V, W and X are not available in this state.

12.1.2 Session 1 State

The Session 1 State occurs following the Open Session operation. Operations in categories C and D are available in this state. Operations in categories A, F, G, H, I, J, K, L, M, N, O, P, Q, R, T, U, V, W and X are not available in this state.

12.1.3 Repository Open 1 State

The Repository Open 1 State occurs once the Session 1 State has been established followed by the Open Repository operation. Operations in categories C, D, F, G, H, I, J and K are available in this state. Operations in categories A, L, M, N, O, P, Q, R, T, U, V, W and X are not available in this state.

12.1.4 SDAI-Model Started RO 1 State

The SDAI-Model Started RO 1 State occurs once the Repository Open 1 State has been established followed by the Start read-only access operation. Operations in categories C, D, F, G, H, I, J, K, L, M, N, O, P, Q and R are available in this state. Operations in categories A, T, U, V, W and X are not available in this state.

12.1.5 SDAI-Model Started RW 1 State

The SDAI-Model Started RW 1 State occurs once the Repository Open 1 State has been established followed by the Start read-write access operation. Operations in categories C, D, F, G, H, I, J, K, L, M, N, O, P, T, U, V, W, and X are available in this state. Operations in category A, Q and R are not available in this state.

12.1.6 State transitions

The transitions from one state to another are specified in table 4. The second through sixth rows represent the initial state, second through sixth columns represent the resulting state and the values in each row for each column specify the operations that cause the transition from the initial to the resulting state. An 'x' value denotes cases where the row and column represent the same state. A '-' value denotes cases where no single operation causes the transition between states.

Table 4 - State transitions for transaction level 1

	No Session 1	Session 1	Repository Open 1	SDAI-Model Started RO 1	SDAI-Model Started RW 1
No Session 1	x	10.3.1	-	-	-
Session 1	10.4.4	x	10.4.5	-	-
Repository Open 1	10.4.4	10.5.3	x	10.7.3	10.7.6
SDAI-Model Started RO 1	10.4.4	10.5.3	10.7.5 10.7.1	x	10.7.4
SDAI-Model Started RW 1	10.4.4	10.5.3	10.7.7 10.7.1	-	x

12.2 State model for transaction level 2

The state model for transaction level 2 implementations consists of five states. These states along with the possible transitions between these states are specified in 12.2.1 through 12.2.6. Operations in categories B and E need not be supported in transaction level 2 implementations.

12.2.1 No Session 2 State

The No Session 2 State occurs before the Open Session operation has been invoked. Operations in category A are available in this state. Operations in categories C, D, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W and X are not available in this state.

12.2.2 Session 2 State

The Session 2 State occurs following the Open Session operation. Operations in categories C and D are available in this state. Operations in categories A, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W and X are not available in this state.

12.2.3 Repository Open 2 State

The Repository Open 2 State occurs once the Session 2 State has been established followed by the Open Repository operation. Operations in categories C, D, F, G, H, I, J and K are available in this state. Operations in categories A, L, M, N, O, P, Q, R, S, T, U, V, W and X are not available in this state.

12.2.4 SDAI-Model Started RO 2 State

The SDAI-Model Started RO 2 State occurs once the Repository Open 2 State has been established followed by the Start read-only access operation. Operations in categories C, D, F, G, H, I, J, K, L, M, N, O, P, Q and R are available in this state. Operations in categories A, S, T, U, V, W and X are not available in this state.

12.2.5 SDAI-Model Started RW 2 State

The SDAI-Model Started RW 2 State occurs once the Repository Open 2 State has been established followed by the Start read-write access operation. Operations in categories C, D, F, G, H, I, J, K, L, M, N, O, P, S, T, U, V, W and X are available in this state. Operations in category A, Q and R are not available in this state.

12.2.6 State transitions

The transitions from one state to another are specified in table 5. The second through sixth rows represent the initial state, the second through sixth columns represent the resulting state and the values in each row for each column specify the operations that cause the transition from the initial to the resulting state. An 'x' value denotes cases where the row and column represent the same state. A '-' value denotes cases where no single operation causes the transition between states.

Table 5 - State transitions for transaction level 2

	No Session 2	Session 2	Repository Open 2	SDAI-Model Started RO 2	SDAI-Model Started RW 2
No Session 2	x	10.3.1	-	-	-
Session 2	10.4.4	x	10.4.5	-	-
Repository Open 2	10.4.4	10.5.3	x	10.7.3	10.7.6
SDAI-Model Started RO 2	10.4.4	10.5.3	10.7.5 10.7.1	x	10.7.4
SDAI-Model Started RW 2	10.4.4	10.5.3	10.7.7 10.7.1	-	x

12.3 State model for transaction level 3

The state model for transaction level 3 implementations consists of ten states. These states along with the possible transitions between these states are specified in 12.1.1 through 12.1.11. Operations in category S need not be supported in transaction level 3 implementations.

12.3.1 No Session 3 State

The No Session 3 State occurs before the Open Session operation has been invoked. Operations in category A are available in this state. Operations in categories B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, T, U, V, W and X are not available in this state.

12.3.2 Session 3 State

The Session 3 State occurs following the Open Session operation. Operations in categories B and C are available in this state. Operations in categories A, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, T, U, V, W and X are not available in this state.

12.3.3 Transaction Started RO 3 State

The Transaction Started RO 3 State occurs once the Session 3 State has been established followed by the Start transaction read-only access operation. Operations in categories C, D and E are available in this state. Operations in categories A, B, F, G, H, I, J, K, L, M, N, O, P, Q, R, T, U, V, W and X are not available in this state.

12.3.4 Transaction Started RW 3 State

The Transaction Started RW 3 State occurs once the Session 3 State has been established followed by the Start transaction read-write access operation. Operations in categories C, D and E are available in this state. Operations in categories A, B, F, G, H, I, J, K, L, M, N, O, P, Q, R, T, U, V, W and X are not available in this state.

12.3.5 Repository Open 3 State

The Repository Open 3 State occurs once the Session 3 State has been established followed by the Open Repository operation. Operations in categories B, C, D and F are available in this state. Operations in categories A, E, G, H, I, J, K, L, M, N, O, P, Q, R, T, U, V, W and X are not available in this state.

12.3.6 RO Repository Open 3 State

The RO Repository Open 3 State occurs once the Transaction Started RO 3 State has been established followed by the Open Repository operation or once the Repository Open 3 State has been established followed by the Start transaction read-only access operation. Operations in categories C, D, E, F, G and H are available in this state. Operations in categories A, B, I, J, K, L, M, N, O, P, Q, R, T, U, V, W and X are not available in this state.