TECHNICAL SPECIFICATION

ISO/TS 10303-27

First edition
2000-11-15

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

Part 27: Implementation methods: Java™ programming language binding to the standard data access interface with Internet/Intranet extensions

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

Partie 27: Méthodes de mise en application: Liaison de langage de programmation Java™ à l'interface d'accès aux données normales avec extensions Internet/Intranet

Reference number
ISO/TS 10303-27:2000(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents

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.

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

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

In other circumstances, particularly when there is an urgent market requirement for such documents, a technical committee may decide to publish other types of normative document:

— an ISO Publicly Available Specification (ISO/PAS) represents an agreement between technical experts in an ISO working group and is accepted for publication if it is approved by more than 50 % of the members of the parent committee casting a vote;

— an ISO Technical Specification (ISO/TS) represents an agreement between the members of a technical committee and is accepted for publication if it is approved by 2/3 of the members of the committee casting a vote.

An ISO/PAS or ISO/TS is reviewed every three years with a view to deciding whether it can be transformed into an International Standard.

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

ISO/TS 10303-27 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *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 methods: The EXPRESS language reference manual;

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

— Part 14, Description methods: The EXPRESS-X language reference manual;

— Part 21, Implementation methods: 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 interface;

— Part 27, Implementation method: Java™ programming language binding to the standard data access interface with Internet/Intranet extensions;

— Part 28, Implementation method: XML representation for EXPRESS-driven data;

— Part 29, Implementation method: Lightweight Java programming language binding to the standard data access interface with Internet/Intranet extensions;

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

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

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

— Part 35, Conformance testing methodology and framework: Abstract test methods for standard data access interface implementations;

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

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

— Part 43, Integrated generic resources: Representation structures;

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

— Part 45, Integrated generic resource: Materials;

— Part 46, Integrated generic resources: Visual presentation;

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

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

— Part 50, Integrated generic resource: Mathematical construct;

— 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 107, Integrated application resource: Engineering analysis core application reference model (EA C-ARM);

— Part 108, Integrated application resource: Parameterization and constraints for explicit geometric product models;

— Part 201, Application protocol: Explicit draughting;

— Part 202, Application protocol: Associative draughting;

— Part 203, Application protocol: Configuration controlled 3D designs of mechanical parts and assemblies;

— 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 mechanical design processes;

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

— Part 223, Application protocol: Exchange of design and manufacturing product information for casting 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 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 233, Application Protocol: Systems engineering data representation;

— Part 234, Application protocol: Ship Operational logs, records, and messages;

— Part 235, Application Protocol: Materials information for the design and verification of products;

— Part 236, Application Protocol: Furniture product data and project data;

— 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 321, Abstract test suite: Functional data and 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 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 333, Abstract test suite: Systems engineering data representation;

— Part 334, Abstract test suite: Ship Operational logs, records, and messages;

— Part 335, Abstract test suite: Materials information for the design and verification of products;

— Part 336, Abstract test suite: Furniture product data and project data;

— 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 presentation;

— Part 519, Application interpreted construct: Geometric tolerances;

— Part 520, Application interpreted construct: Associative draughting elements;

— Part 1001, Application module: Appearance assignment;

— Part 1002, Application Module: Colour;

— Part 1003, Application module: Curve appearance;

— Part 1004, Application module: Elemental shape;

— Part 1005, Application module: Elemental topological shape;

— Part 1006, Application module: Foundation representation;

— Part 1007, Application module: General surface appearance;

— Part 1008, Application Module: Layer assignment;

— Part 1009, Application Module: Shape appearance and layers;

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 14 specify the description methods,

— Parts 21 to 29 specify the implementation methods,

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

— Parts 41 to 50 specify the integrated generic resources,

— Parts 101 to 108 specify the integrated application resources,

— Parts 201 to 236 specify the application protocols,

— Parts 301 to 336 specify the abstract test suites,

— Parts 501 to 520 specify the application interpreted constructs,

— Parts 1001 to 1009 specify the application module

Should further parts of ISO 10303 be published, they will follow the same numbering pattern.

Annexes A and B form a normative part of this part of ISO 10303. Annexes C to 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 organised 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 integrated 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 complete Java™ programming language binding to the standard data access interface (SDAI). SDAI is the access interface specification to data that has been defined using ISO 10303-11. SDAI is specified in ISO 10303-22.

Major subdivisions in this part of ISO 10303 are:

— Fundamental concepts in clauses 4 and 5;

— SDAI session classes to manage the environment in clause 6;

— Late binding: Generic Java interfaces and classes for arbitrary EXPRESS schemas in clause 7;

— Early binding: Specific Java interfaces and classes for particular EXPRESS schemas in clause 8;

— Error handling through the Java exception mechanism in clause 9.

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

Part 27:

Implementation methods: Java™ programming language binding to the standard data access interface with Internet/Intranet extensions

1 Scope

This part of ISO 10303 specifies a binding of the Java[1)] programming language to application data modelled in EXPRESS, ISO 10303-11 and to the standard data access interface, ISO 10303-22. It also specifies an import and export mechanism for data according to the clear text encoding of the exchange structure, ISO 10303-21. A further extension is that SDAI repositories can be created, deleted and linked while the SDAI session is open. Dynamically linking SDAI repositories through a network like Internet or Intranet allows accessing and changing of remote data.

In addition to the scope of ISO 10303-22 the scope of this part of ISO 10303 contains:

— creating, deletion and linking of the data repositories during an SDAI session;

— specific support for linking a remote data repository through a network like Internet or Intranet;

— convenience interfaces, classes, fields and methods suitable to this Java programming language binding;

— implementation mechanisms for the handling of errors as specified in ISO 10303-22;

— import from and export to the clear text encoding of the exchange structure as specified in ISO 10303-21.

1) Java is the trade mark of a product supplied by Sun Microsystems, Inc. This information is given for the convenience of users of this part of ISO 10303 and does not constitute an endorsement by ISO of the product name. Equivalent products may be used if they can be shown to lead to the same result.

2 Normative references

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

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

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

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

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

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

Technical Corrigendum 1 to 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*.

The Java Language Specification, Second Edition by James Gosling, Bill Joy and Guy Steele, Addison Wesley, Reading Massachusetts, 2000, ISBN: 0201310082.
http://java.sun.com/docs/books/jls/index.html

The Java Virtual Machine Specification (2nd edition) by Tim Lindhold and Frank Yellin, Addison Wesley, Reading Massachusetts, 1999, ISBN: 0201432943.
http://java.sun.com/docs/books/vmspec/index.html

3 Terms, definitions, and abbreviations

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

3.1 Terms defined in ISO 10303-1

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

— application;

— application protocol;

— data;

— implementation method;

— information;

— information model;

— product information model;

— structure.

3.2 Terms defined in ISO 10303-11

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

— aggregation data type;

— complex entity data type;

— partial complex entity data type;

— constructed data type;

— data type;

— defined data type;

— entity;

— entity data type;

— entity instance;

— enumeration data type;

— instance;

— named data type;

— population;

— select data type;

— value.

3.3 Terms defined in ISO 10303-22

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

— application schema;

— constraint;

— current schema;

— external schema;

— foreign schema;

— identifier;

— implementation class;

— iterator;

— native schema;

— repository;

— schema instance;

— SDAI-model;

— session;

— validation.

3.4 Terms defined in The Java Language Specification

For the purposes of this part of ISO 10303, the following terms defined in The Java Language Specification apply:

— class;

— constructor;

— exception;

— field;

— interface;

— member;

— method;

— overloading;

— package;

— parameter;

— primitive;

— reference;

— statement;

— type.

3.5 Other definitions

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

3.5.1 aggregate class: a Java class representing an aggregation data type.

3.5.2 aggregate-dependent: The base type of an entity attribute or an aggregation type is **aggregate-dependent** if the aggregate value, returned by the EXPRESS **TypeOf** function for this base type contains the name of a defined type whose underlying type is an aggregation data type or if the base type is directly an aggregation type.

3.5.3 application entity: an entity declared in an application schema.

3.5.4 dictionary entity: an entity declared in the `sdai_dictionary_schema`.

3.5.5 entity class: a Java class representing a complex or single entity data type of an application schema or the `sdai_dictionary_schema`.

3.5.6 entity interface: a Java interface representing an entity data type of an application schema or the `sdai_dictionary_schema`.

3.5.7 persistent label: a string by which an entity instance can be uniquely identified within a SDAI repository.

3.5.8 receiver: a Java object upon which a specified member method is invoked.

3.5.9 SDAI method: a Java method representing an SDAI operation.

3.5.10 select-dependent: The base type of an entity attribute or an aggregation type is **select-dependent** if the aggregate value, returned by the EXPRESS **TypeOf** function for this base type contains the name of a defined type whose underlying type is a SELECT data type.

3.5.11 select path: a sequence of defined data types needed to specify a value of a select data type as defined in the technical corrigendum of ISO 10303-21:11.1.8.

3.5.12 session class: a specialized Java class for a session entity.

3.5.13 session entity: an entity declared in the `sdai_session_schema` or the `sdai_population_schema`.

3.6 Abbreviations

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

SDAI Standard Data Access Interface, ISO 10303-22

part-21 Clear text encoding of the exchange structure, ISO 10303-21

URL Universal Resource Locator

4 General characteristics

ISO 10303-22 specifies operations independently of any programming language. Language bindings of the operations are developed for programming languages to define the functionality required of conforming implementations. Two types of language bindings are identified: late bindings and early bindings. The concept of language bindings is defined in ISO 10303-22 clause 4.8. This part of ISO 10303 specifies an harmonised late and early language binding of Java to the SDAI. For this, Java packages, classes, interfaces, methods and data fields implementing SDAI and EXPRESS elements are defined. This part of ISO 10303 extends the functionality defined in ISO 10303-22 to provide more efficient or convenient operations.

Entity data types are mapped to the Java interface `EEntity` and other interfaces extending it for early binding access. These interfaces are called **entity interfaces**. Complex entity data types are mapped to the Java class `CEntity` and other classes extending it for early binding access. These classes are called **entity classes**. Each entity class implements one or several entity interfaces. Aggregation data types are mapped to the interface Aggregate and specialised classes implementing it. The class `AEntity` and other classes extending it represent aggregations of entity instances only. Further Java interfaces and classes are specified for other EXPRESS elements like schema and defined data types.

The SDAI operations are mapped to public Java methods. The input and output parameters of an SDAI operation are mapped to the parameters, receiver and return value of the corresponding method in Java. These methods are called **SDAI methods**. While executing SDAI methods errors may occur. These errors are reported to the application program by throwing an SdaiException. Every SDAI method throws SdaiException. The possible error indicators for an SDAI method are specified in the definition of the corresponding SDAI operation and, in addition, in this part of ISO 10303.

> Note - An implementation may have further public or protected data fields and methods, however, to ensure robustness, such additional public or protected data fields and methods should be avoided. In the case where they cannot be avoided, their usage should be specified by the implementation. Except for classes representing aggregates all classes defined here should have no public constructors.

The entities defined in the `SDAI_session_schema` and the `SDAI_population_schema` are not mapped to entity classes and entity interfaces. Instead, specific session classes are defined for them, see clause 6. The `SDAI_dictionary_schema` schema is mapped identically to application schemas with a few exceptions as defined in clause 10. The mapping of the `SDAI_parameter_data_schema` to Java classes and interfaces is defined in clause 5.

An implementation of this part of ISO 10303 is free to substitute any specified Java interface by an equivalent Java class as long as the Java class behaves exactly as the specified Java interface from a user perspective.

An application program should not extend or implement any of the Java interfaces or Java classes specified in this part of ISO 10303.

4.1 Entity Attribute and Aggregate Member access

Early and late binding SDAI methods are defined for the following entity and aggregate operations:

— `Test attribute`, specified in ISO 10303-22:10.10.2;

— `Get attribute`, specified in ISO 10303-22:10.10.1;

— `Put attribute`, specified in ISO 10303-22:10.11.3;

— `Unset attribute value`, specified in ISO 10303-22:10.11.4;

— `Create aggregate instance`, specified in ISO 10303-22:10.11.5;

— `Test current member`, specified in ISO 10303-22:10.17.2;

— `Test by index`, specified in ISO 10303-22:10.17.1;

— `Is member`, specified in ISO 10303-22:10.12.2;

— `Get by index`, specified in ISO 10303-22:10.15.1;

— `Get current member`, specified in ISO 10303-22:10.12.7;

— `Put by index`, specified in ISO 10303-22:10.16.1;

— `Put current member`, specified in ISO 10303-22:10.13.2;

— `Add before current member`, specified in ISO 10303-22:10.19.1;

— `Add after current member`, specified in ISO 10303-22:10.19.2;

— `Add by index`, specified in ISO 10303-22:10.19.3;

— `Add unordered`, specified in ISO 10303-22:10.14.1;

— `Unset value by index`, specified in ISO 10303-22:10.18.1;

— `Unset value current, member` specified in ISO 10303-22:10.18.2;

— `Create aggregate instance by index`, specified in ISO 10303-22:10.16.2;

— `Create aggregate instance as current member`, specified in ISO 10303-22:10.13.1;

— `Create aggregate instance before current member`, specified in ISO10303-22:10.19.4;

— `Create aggregate instance after current member`, specified in ISO10303-22:10.19.5;

— `Add aggregate instance by index`, specified in ISO 10303-22:10.19.6;

— `Create aggregate instance unordered`, specified in ISO 10303-22:10.14.2;

— `Remove by index`, specified in ISO 10303-22:10.19.7;

— `Remove unordered`, specified in ISO 10303-22:10.14.3;

— `Remove current member`, specified in ISO 10303-22:10.13.3.

An SdaiException VA_NSET shall be thrown when `Get attribute`, `Get by index`, or `Get current member` operation would result in an unset value. The `Test attribute`, `Test by index`, or `Test current member` operation shall be used to check for unset values.

An SdaiException VT_NVLD shall be thrown when `Put Attribute`, `Put by index`, `Put current member`, `Add before current member`, `Add after current member`, `Add by index`, and `Add unordered` is used with an unset value. The operations `Unset attribute value`, `Unset value by index`, or `Unset value current member` shall be used to unset values.

An SdaiException VT_NVLD shall be thrown when `Put Attribute`, `Put by index`, `Put current member`, `Add before current member`, `Add after current member`, `Add by index`, and `Add unordered` is applied with a value of type aggregate. The operations `Create aggregate instance`, `Create aggregate instance by index`, `Create aggregate instance as current member`, `Create aggregate instance before current member`, `Create aggregate instance after current member`, `Create aggregate instance unordered`, and `Add aggregate instance by index` shall be used to create new aggregates.

Attributes of session entities are read-only. They can only be accessed through the specialized early binding style methods defined in clause 6.

For late and early binding it is important whether or not the base type of an entity attribute or an aggregation type is **select-dependent**.

4.2 Dynamic SdaiRepositories

While an `SdaiSession` is open all the available `SdaiRepositories` for this `SdaiSession` are accessible through the attribute `known_servers`. SDAI defines no operations to dynamically create, delete, or link other SdaiRepositories to the SdaiSession. This language binding defines additional SDAI methods to create, delete, link, and unlink `SdaiRepositories` while a `SdaiSession` is open. Furthermore, import and export functionality to part-21 exchange structures is defined.

After the `open session` operation, an initial, implementation dependent set known_servers is available. Further SdaiRepositories can be linked to the SdaiSession with the `linkRepository` method, see 6.1.6. For this, the additional attribute `location` is introduced for SdaiRepositories.

During `linkRepository` `location` is used to specify where to find the SdaiRepository. The interpretation of `location` is implementation dependent. It may be a path in the local file system or an Internet/Intranet address with login information such as user and password. Implementations may also support specific repository names from which the location of a repository can be derived.

A linked SdaiRepository may contain entity instances referencing other entity instances in other SdaiRepositories. These referenced SdaiRepositories shall also be included in the set of known_servers. However, the location of these SdaiRepositories may be unknown - such repositories are called **degenerated**. Degenerated SdaiRepositories can't be opened. The getLocation method applied to such an SdaiRepository returns a null value. Only the name of a degenerated SdaiRepository is known. The linkRepository method can be used to specify a location for a degenerated SdaiRepository and to find an SdaiRepository in the set of known_servers by name.

In a distributed environment it may happen that there are several SdaiRepositories with the same name. However, the name in the set of known_servers must be unique. To solve this problem, an application may remove the location information from an SdaiRepository with the unlinkRepository method, see 6.3.9, and then link it to a new location.

Note: The methods linkRepository and unlinkRepository do not copy the contents of the SdaiRepositories. They only influence the connection between the SdaiSession and the physical locations of SdaiRepositories.

New SdaiRepositories can be created with the create method, see 6.1.7. The `deleteRepository` operation physically deletes an SdaiRepository, see 6.3.8. A part-21 exchange structure can be imported into a new SdaiRepository with the importClearTextEncoding method, see 6.1.8.

The creation, deletion, linking and unlinking of SdaiRepositories, and the importClearTextEncoding are operations outside the SdaiTransaction mechanism. Applications may commit or abort any pending SdaiTransaction before invoking one of these operations. It is not possible to undo or abort these operations afterwards.

4.3 Packages and EXPRESS schema

Throughout this document the package prefix "SDAI." is used as a placeholder for an implementation specific package name. Implementations shall not directly use the package prefix "SDAI.".

Note 1 - *The Java Language Specification* provides guidance how to choose unique package names.

All schema-independent Java interfaces and classes defined in this part of ISO 10303 shall be contained in a Java package named `SDAI.lang`. The SDAI_dictionary_schema schema is mapped into package `SDAI.dictionary`.

Further early binding packages are defined in clause 8.2.

4.4 Aggregates

The EXPRESS aggregates LIST, ARRAY, SET, BAG and also the SDAI non_persistent_list are treated as Aggregates, see 7.4. However, instances of the aggregates need some knowledge of the purpose for which they are used. For this, the convenience method `getAggregationType`, see

7.4.1, is introduced. It is the responsibility of the implementation to allow only such operations on the aggregates, which fit to the aggregation type. Otherwise, an SdaiException has to be thrown.

4.5 Conformance requirements

An implementation of this part of ISO 10303 shall conform to an implementation class as specified in ISO 10303-22 clause 13. The level of session event recording support, specified in ISO 10303-22:13.1.3 and the level of scope support, specified in ISO 10303-22:13.1.4 shall not be considered when determining the implementation class, specified in ISO 10303-22:13.2.

For this language binding support of the `SDAI_dictionary_schema` for late binding access is mandatory.

The implementation shall support all SDAI methods whose original specification in ISO 10303-22 contains an operation required by the implementation class and shall support all convenience functions defined in this part of ISO 10303.

This part of ISO 10303 does not define details on how to implement the specified Java classes and methods.

4.6 Agglomerate domain

As defined in ISO 10303-22:8.4.1, a schema_instance is a logical collection of sdai_models, defining the **domain** over which references between entity instances in different SdaiModels are supported. The following SDAI operations use an aggregate of schema_instances as an **agglomerate domain** of schema_instances:

— `Find entity instance users`, specified in ISO 10303-22:10.10.8;

— `Find entity instance usedin`, specified in ISO 10303-22:10.10.9;

— `Find instance roles`, specified in ISO 10303-22:10.10.11.

Furthermore, such an agglomerate domain is needed for the operation Get Attribute, specified in ISO 10303-22:10.10.1, in the case of derived and inverse attributes. For the purpose of this language binding to the SDAI, an agglomerate domain shall be represented by an aggregate of sdai_models, ASdaiModel. This ASdaiModel aggregate is defined as a union of sets **associated_models** taken over all members of the aggregate of schema_instances. The ASchemaInstance method getAssociatedModels, see clause 6.7.1, can be used to retrieve the agglomerate domain of SdaiModels.

4.7 Date and time

In Java, the actual date and time can be retrieved with the `java.lang.System` method `currentTimeMillis` as a Java `long` value with a precision of a millisecond. This `long` value has a higher accuracy than the date and time specification defined by the string-type time_stamp, see ISO 10303-22:7.3.3. Therefore, all validation and change times in this language binding can be alternatively accessed as values of type long.

5 Mapping of the SDAI_parameter_data_schema

This clause defines how EXPRESS types from the **SDAI_parameter_data_schema**, specified in ISO 10303-22 clause 9, shall be mapped to the Java programming language.

5.1 String_value

The Java class `java.lang.String` shall be associated with the SDAI data type **string_value**. For late binding attribute access the value shall be passed as an instance of type `java.lang.Object.`

5.2 Binary_value

The Java class `SDAI.lang.Binary` shall be associated with the SDAI data type **binary_value**. For late binding attribute access the value shall be passed as an instance of type `java.lang.Object.`

```
package SDAI.lang;

public final class Binary {
    public Binary(String str) throws SdaiException;
    public Binary(byte [] str, int size) throws SdaiException;
    public String toString();
    public int toByteArray(byte [] byte_array);
    public int getSize();
    boolean equals(Binary bin);
}
```

5.3 Integer_value

By default, the Java type `int` shall be associated with the SDAI data type **integer_value**. Late binding implementations may represent the actual value using an instance of `java.lang.Integer` or `java.math.BigInteger`, and the value shall be passed as an instance of type `java.lang.Object.`

5.4 Real_value

By default, the Java type `double` shall be associated with the SDAI data type **real_value**. Late binding implementations may represent the actual value using an instance of `java.lang.Double` or `java.math.BigDecimal`, and the value shall be passed as an instance of type `java.lang.Object`.

5.5 Number_value

By default, the Java type `double` shall be associated with the SDAI data type **number_value**. Late binding implementations may represent the actual value using an instance of `java.lang.Double` or `java.math.BigDecimal`, and the value shall be passed as an instance of type `java.lang.Object`.

5.6 Boolean_value

The Java type `boolean` shall be associated with the SDAI data type **boolean_value**.

5.7 Logical_value

The Java type `int` shall be associated with the SDAI data type **logical_value**.

5.8 Query_source

Query_source shall be associated with a method, `query`, in `Aggregate`, `SdaiModel`, `SdaiRepository`, and `SchemaInstance`.

5.9 Iterator

The Java class type `SdaiIterator` shall be associated with the SDAI data type **iterator**.

5.10 Entity_instance

A Java object of type `EEntity` shall represent either an **application_instance** or **dictionary_instance**. A Java object of type `SdaiSession`, `SdaiRepository`, `SchemaInstance`, `SdaiModel`, `SdaiTransaction`, or `EntityExtent` shall represent **session_instance**.

5.11 Attribute_value

This part of ISO 10303 does not define how to store **attribute_values**, however, it defines methods for late- and early-binding access to attribute_values.

5.12 Select_value

This part of ISO 10303 does not define how to store **select_values**, however, it defines how select_values are handled for late and early-binding entity attribute access.

Select_values shall be handled differently for late and early binding attribute access. For both bindings the same rules as for Select data types, defined in the technical corrigendum as specified in ISO 10303-21:11.1.8, shall be used.

For late binding attribute access the actual select type information for the attribute access methods shall be defined through a parameter of type Java array of EDefined_type.

For early binding attribute access the actual select type information shall be defined through zero, one, or more parameters of the early binding Java interface types for the attribute access methods.

5.13 Select_aggregate_instance

This part of ISO 10303 does not define how to store `select_aggregate_instances`.

5.14 Enumeration_value

An `enumeration_value` shall be mapped to the Java type `int`. The numbering of the enumeration values shall correspond to the list indices of attribute `elements` of SDAI entity **enumeration_type**, specified in ISO 10303-22:6.4.28.

5.15 Aggregate_instance

An `aggregate_instance` shall be mapped to an instance of the Java interface `Aggregate` or Java interfaces which extend this interface.

The following subtypes of `aggregate_instance` shall be mapped to instances of `Aggregate`:

— set_instance;

— bag_instance;

— schema_defined_list_instance;

— non_persistent_list_instance;

— array_instance;

— application_indexed_array_instance.

Except for `non_persistent_list_instances`, an application shall not use the Java `new` operator to create instances of any class implementing `Aggregate`.

For late binding attribute access the value shall be passed as an instance of type `java.lang.Object.`

5.16 Non_persistent_list_instance

A `non_persistent_list_instance` shall be either mapped to an instance of the Java class `AEntity` or to an instance of an early binding aggregate, see clause 8.10. Non persistent lists are created with the Java `new` operator.

> Note - There is no way to use an instance of a non_persistent_list as a value for an entity attribute. Non persistent lists are used only temporarily.

6 SDAI Session classes

This clause describes the session classes that represent the session entities, specified in ISO 10303-22:7 and ISO 10303-22:8 and aggregate classes thereof.

6.1 SdaiSession

The `SdaiSession` class shall represent the SDAI entity **sdai_session**, specified in ISO 10303-22:7.4.1.

In SDAI the inverse attribute **active_transaction** is defined as **SET [0:1] of sdai_transaction**, see ISO 10303-22:7.4.1. For the purpose of this language binding **active_transaction** is redefined as **OPTIONAL sdai_transaction**.

```
package SDAI.lang;
import SDAI.dictionary.*;
public class SdaiSession {
  public Implementation getSdaiImplementation() throws SdaiException;
  public ASdaiRepository getKnownServers() throws SdaiException;
  public ASdaiRepository getActiveServers() throws SdaiException;
  public boolean testDataDictionary() throws SdaiException;
  public SchemaInstance getDataDictionary() throws SdaiException;
  public ASdaiModel getActiveModels() throws SdaiException;
  public boolean testActiveTransaction() throws SdaiException;
  public SdaiTransaction getActiveTransaction() throws SdaiException;

  public static SdaiSession openSession()throws SdaiException;
  public void closeSession() throws SdaiException;
  public SdaiTransaction startTransactionReadWriteAccess() throws SdaiException;
  public SdaiTransaction startTransactionReadOnlyAccess() throws SdaiException;

  public EEntity_definition getComplexEntityDefinition(EEntity_definition types[],
    ESchema_definition schema) throws SdaiException;

  public Class getComplexEntityClass(Class[] types, Class schema) throws SdaiException;

  public SdaiRepository linkRepository(String repository_name, Object location)
    throws SdaiException;
  public SdaiRepository createRepository(String repository_name, Object location)
    throws SdaiException;
  public SdaiRepository importClearTextEncoding (String repository_name, Object source_location,
    Object destination_location) throws SdaiException;
}
```

6.1.1 Open session

The static method `openSession` shall implement the SDAI operation **Open session**, specified in ISO 10303-22:10.3.1.

6.1.2 Close session

The `closeSession` method shall implement the SDAI operation **Close session**, specified in ISO 10303-22:10.4.4.

6.1.3 Start transaction read-write access

The `startTransactionReadWriteAccess` method shall implement the SDAI operation **Start transaction read-write access**, specified in ISO 10303-22:10.4.6.

6.1.4 Start transaction read-only access

The `startTransactionReadOnlyAccess` method shall implement the SDAI operation **Start transaction read-only access**, specified in ISO 10303-22:10.4.7.

6.1.5 Get complex entity definition

The `getComplexEntityDefinition and getComplexEntityClass` methods shall implement the SDAI operation **Get complex entity definition**, specified in ISO 10303-22:10.9.1.

The `types` array shall contain the leaf entity data types of the complex entity data type. Two overloaded methods are specified:

— the late binding method accepts and returns EEntity_`definitions`;

— the early binding method accepts Java classes for entity interfaces and returns the entity class representing complex entity data type. This class shall implement all entity interfaces specified by the `types` parameter.

NOTE - The return value may be used as an input parameter for `Create entity instance`.

6.1.6 Link repository

The `linkRepository` method is a convenience method to make an already existing but external repository known to the current session. On success, the external repository shall be added to the current session's `known_servers` and returned. All repositories which are referenced by the application_instances or schema_instances, associated with this repository, shall also be added to the set of `known_servers`. This method is an extension to the functionality of SDAI.

The parameter `location` shall be used to specify an address of the repository. The interpretation of this parameter is implementation-dependent. In the case when the value of this parameter is null, the method searches for the repository with a given name within `known_servers` and, if success, returns it.

Possible error indicators:
SS_NOPN An SDAI session is not open.
TR_NEXS A transaction has not been started.
RP_NEXS The repository does not exist.
RP_DUP Repository duplicate.
SY_ERR An underlying system error occurred.

6.1.7 Create repository

The `createRepository` method is a convenience method which shall be used to create a repository.

The parameter `repository_name` shall specify the name of the SDAI repository. The parameter `location` shall be used to specify the address of the newly created repository. The value of this parameter is implementation-dependent. The value may be `null`. The created repository is added to the list of known_servers.

Possible error indicators:
SS_NOPN An SDAI session is not open.
RP_DUP Repository duplicate.
SY_ERR An underlying system error occurred.

6.1.8 Import clear text encoding

The `importClearTextEncoding` method shall be used to create a new repository, then to import a part-21 exchange structure into this repository, and open this repository. On success, the new repository is added to the set of known_servers and also to the set of **active_servers**.

The parameter `name` specifies the name of the SDAI repository. If this parameter is `null` the value of the attribute `name` in the header entity `file_name`, as defined in ISO 10303-21:9.2.2, shall be used as the name for the SDAI repository. If the name is invalid, then the SdaiException VT_NVLD is thrown.

The parameter `source_location` is used to specify where to find the exchange structure. The interpretation of this parameter is implementation-dependent.

The parameter `destination_location` is used to specify where to place the new created repository. The interpretation of this parameter is implementation-dependent. The value of this parameter may be `null`.

Possible error indicators:
SS_NOPN An SDAI session is not open.
FN_NAVL This function is not supported by this implementation.
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.
VT_NVLD The SDAI repository name is invalid.
RP_DUP Repository duplicate.
SY_ERR An underlying system error occurred.

6.2 Implementation

The `Implementation` class shall represent the SDAI entity **implementation**, specified in ISO 10303-22:7.4.2

```
package SDAI.lang;
```

```
import SDAI.dictionary.*;

public class Implementation {
  public String getName() throws SdaiException;
  public String getLevel() throws SdaiException;
  public String getSdaiVersion() throws SdaiException;
  public String getBindingVersion() throws SdaiException;
  public int getImplementationClass() throws SdaiException;
  public int getTransactionLevel() throws SdaiException;
  public int getExpressionLevel() throws SdaiException;
  public int getDomainEquivalenceLevel() throws SdaiException;
}
```

Note - The **level of event recording** and the **level of scope support** are not available for this language binding.

6.3 SdaiRepository

The `SdaiRepository` class shall represent the SDAI entity **sdai_repository**, specified in ISO 10303-22:7.4.3 and the SDAI entity **sdai_repository_contents**, specified in ISO 10303-22:7.4.4.

```
package SDAI.lang;
import SDAI.dictionary.*;

public class SdaiRepository {
  public String getName() throws SdaiException;
  public Object getLocation()throws SdaiException;
  public A_string getDescription() throws SdaiException;
  public SdaiSession getSession() throws SdaiException;
  public ASdaiModel getModels() throws SdaiException;
  public ASchemaInstance getSchemas() throws SdaiException;
```

Further convenience methods are specified to access the attributes of the header entity `file_name`, see ISO 10303-21:

```
  public String getChangeDate() throws SdaiException;
  public long getChangeDateLong() throws SdaiException;
  public A_string getAuthor() throws SdaiException;
  public A_string getOrganization() throws SdaiException;
  public String getPreprocessorVersion() throws SdaiException;
  public void setPreprocessorVersion(String value) throws SdaiException;
  public String getOriginatingSystem() throws SdaiException;
  public void setOriginatingSystem(String value) throws SdaiException;
  public String getAuthorization() throws SdaiException;
  public void setAuthorization(String value) throws SdaiException;

  //For early binding:
  public SdaiModel createSdaiModel(String model_name, Class schema) throws SdaiException;
  //For late binding:
  public SdaiModel createSdaiModel(String model_name, ESchema_definition schema)
    throws SdaiException;
  //For early binding:
  public SchemaInstance createSchemaInstance(String name, Class schema) throws SdaiException;
```

```
  //For late binding:
  public SchemaInstance createSchemaInstance(String name,
    ESchema_definition schema) throws SdaiException;
  public void openRepository() throws SdaiException;
  public void closeRepository() throws SdaiException;
  public EEntity getSessionIdentifier(String Label) throws SdaiException;
  public int query(String where, EEntity entity, AEntity result) throws SdaiException;
  public void exportClearTextEncoding(Object location) throws SdaiException;
  public void deleteRepository() throws SdaiException;
  public void unlinkRepository() throws SdaiException;
}
```

Note - Depending on the result of the ongoing ISO 10303-21 standardization the following methods may be added in future to support `section_language` and `section_context` :

```
public A_string getContextIdentifiers() throws SdaiException;
public String getDefaultLanguage() throws SdaiException;
public void setDefaultLanguage(String value) throws SdaiException;
```

6.3.1 Create SDAI-model

The `createSdaiModel` method shall implement the SDAI operation `Create SDAI-model`, specified in ISO 10303-22:10.5.1.

6.3.2 Create schema instance

The `createSchemaInstance` method shall implement the SDAI operation `Create schema instance`, specified in ISO 10303-22:10.5.2.

6.3.3 Open repository

The `openRepository` method shall implement the SDAI operation `Open repository`, specified in ISO 10303-22:10.4.5.

6.3.4 Close repository

The `closeRepository` method shall implement the SDAI operation `Close repository`, specified in ISO 10303-22:10.5.3.

6.3.5 Get session identifier

The `getSessionIdentifier` method shall implement the SDAI operation `Get session identifier`, specified in ISO 10303-22:10.11.7.

6.3.6 SDAI query

The `query` method shall implement the SDAI operation `SDAI query`, specified in ISO 10303-22:10.4.14, in the case that the `query_source` shall be an SDAI-repository.

6.3.7 Export clear text encoding

The `exportClearTextEncoding` method shall export an SDAI repository and its contents into a new instance of a clear text encoding as defined in ISO 10303-21.

The parameter `location` specify where to place the resulting part-21 exchange structure. The interpretation of this parameter is implementation-dependent.

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

6.3.8 Delete repository

The `deleteRepository` method shall delete an `SdaiRepository` along with all its `SdaiModels` and `SchemaInstances`. This is done by first invoking the operation `delete schema instance`, specified in ISO 10303-22:10.6.1, on every contained `SchemaInstance`, then, by invoking the operation `delete SDAI-model` on every contained `SdaiModel`, and at last, by removing the SdaiRepository from the set of `known_servers` in `SdaiSession`. If open, the SDAI repository shall be closed by the implementation.

The repository is deleted physically and cannot be linked during this or future sessions.

Possible error indicators:
SS_NOPN An SDAI session is not open.
FN_NAVL This function is not supported by this implementation.
TR_RW The transaction is read-write and changes are unresolved.
RP_NAVL The repository is currently not available.
RP_NEXS The repository does not exist
SY_ERR An underlying system error occurred.

6.3.9 Unlink repository

The `unlinkRepository` method shall remove an SDAI repository from the current session.

Possible error indicators:
SS_NOPN An SDAI session is not open.
FN_NAVL This function is not supported by this implementation.
RP_NEXS The repository does not exist.
RP_OPN The repository is open.
SY_ERR An underlying system error occurred.

6.4 ASdaiRepository

The `ASdaiRepository` class shall represent an aggregate of the SDAI entity **sdai_repository**, specified in ISO 10303-22:7.4.3.

```
package SDAI.lang;

public class ASdaiRepository implements Aggregate {
  public ASdaiRepository();
  public boolean isMember(SdaiRepository value) throws SdaiException;
  public SdaiRepository getByIndex(int index) throws SdaiException;
  public SdaiRepository getCurrentMember(SdaiIterator iter) throws SdaiException;
}
```

6.5 SdaiTransaction

The `SdaiTransaction` class shall represent the SDAI entity **sdai_transaction**, specified in ISO 10303-22:7.4.5.

```
package SDAI.lang;

public class SdaiTransaction {
  public static final int NO_ACCESS;
  public static final int READ_ONLY;
  public static final int READ_WRITE;
  public int getMode() throws SdaiException;
  public SdaiSession getOwningSession() throws SdaiException;
  public void commit() throws SdaiException;
  public void abort() throws SdaiException;
  public void endTransactionAccessCommit() throws SdaiException;
  public void endTransactionAccessAbort() throws SdaiException;
}
```

6.5.1 Commit

The `commit` method shall implement the SDAI operation `Commit`, specified in ISO 10303-22:10.4.8.

6.5.2 Abort

The `abort` method shall implement the SDAI operation `Abort`, specified in ISO 10303-22:10.4.9.

6.5.3 End transaction access and commit

The `endTransactionAccessCommit` method shall implement the SDAI operation `End transaction access and commit`, specified in ISO 10303-22:10.4.10.

6.5.4 End transaction access and abort

The `endTransactionAccessAbort` method shall implement the SDAI operation `End transaction access and abort`, specified in ISO 10303-22:10.4.11.

6.6 SchemaInstance

The `SchemaInstance` class shall represent the SDAI entity **schema_instance**, specified in ISO 10303-22:8.4.1

```
package SDAI.lang;
import SDAI.dictionary.*;

public class SchemaInstance {
  public String getName() throws SdaiException;
  public ASdaiModel getAssociatedModels() throws SdaiException;
  public ESchema_definition getNativeSchema() throws SdaiException;
  public String getNativeSchemaString() throws SdaiException;
  public SdaiRepository getRepository() throws SdaiException;
  public boolean testChangeDate() throws SdaiException;
  public String getChangeDate() throws SdaiException;
  public long getChangeDateLong() throws SdaiException;
  public String getValidationDate() throws SdaiException;
  public long getValidationDateLong() throws SdaiException;
  public int getValidationResult() throws SdaiException;
  public int getValidationLevel() throws SdaiException;

  public void delete() throws SdaiException;
  public void rename(String name) throws SdaiException;
  public void addSdaiModel(SdaiModel model) throws SdaiException;
  public void removeSdaiModel(SdaiModel model) throws SdaiException;
  public int query(String where, EEntity entity, AEntity result) throws SdaiException;
  public int validateGlobalRule(EGlobal_rule rule, AEntity nonConf) throws SdaiException;
  public int validateGlobalRule(String rule, AEntity nonConf) throws SdaiException;
  public int validateUniquenessRule(EUniqueness_rule rule, AEntity nonConf) throws SdaiException;
  public int validateUniquenessRule(String rule, AEntity nonConf) throws SdaiException;
  public int validateInstanceReferenceDomain(AEntity nonConf) throws SdaiException;
  public int validateSchemaInstance() throws SdaiException;
  public boolean isValidationCurrent() throws SdaiException;
}
```

6.6.1 getValidationDateLong

The `getValidationDateLong` method is a convenience method which shall return the Java `long` equivalent of **validation_date** of **schema_instance**, see ISO 10303-22:8.4.1.

6.6.2 Delete schema instance

The `delete` method shall implement the SDAI operation `Delete schema instance`, specified in ISO 10303-22:10.6.1.

6.6.3 Rename schema instance

The `rename` method shall implement the SDAI operation `Rename schema instance`, specified in ISO 10303-22:10.6.2.

6.6.4 Add SDAI-model

The `addSdaiModel` method shall implement the SDAI operation `Add SDAI-model`, specified in ISO 10303-22:10.6.3.

6.6.5 Remove SDAI-model

The `removeSdaiModel` method shall implement the SDAI operation `Remove SDAI-model`, specified in ISO 10303-22:10.6.4.

6.6.6 SDAI query

The `query` method shall implement the SDAI operation `SDAI query`, specified in ISO 10303-22:10.4.14 in the case that the `query_source` is a schema instance.

6.6.7 Validate global rule

The `validateGlobalRule` methods shall implement the SDAI operation `Validate global rule`, specified in ISO 10303-22:10.6.5.

6.6.8 Validate uniqueness rule

The `validateUniquenessRule` methods shall implement the SDAI operation `Validate uniqueness rule`, specified in ISO 10303-22:10.6.6.

6.6.9 Validate instance reference domain

The `validateInstanceReferenceDomain` method shall implement the SDAI operation `Validate instance reference domain`, specified in ISO 10303-22:10.6.7.

6.6.10 Validate schema instance

The `validateSchemaInstance` method shall implement the SDAI operation `Validate schema instance`, specified in ISO 10303-22:10.6.8.

6.6.11 Is validation current

The `isValidationCurrent` method shall implement the SDAI operation `Is validation current`, specified in ISO 10303-22:10.6.9.

6.7 ASchemaInstance

The `ASchemaInstance` class shall represent an aggregate of the SDAI entity **schema_instance**, specified in ISO 10303-22:8.4.1.

```
package SDAI.lang;

public class ASchemaInstance implements Aggregate {
  public ASchemaInstance();
```

```
  public boolean isMember(SchemaInstance value) throws SdaiException;
  public SchemaInstance getCurrentMember(SdaiIterator iter) throws SdaiException;
  public SchemaInstance getByIndex(int index) throws SdaiException;
  public void addByIndex(int index, SchemaInstance value) throws SdaiException;
  public ASdaiModel getAssociatedModels()throws SdaiException;
}
```

NOTE - The usage of `ASchemaInstance` for `sdai_repository_contents.schemas` and `sdai_model.associated_with` is read-only. But ASchemaInstance can also be used as a specialized non-persistent list for operations like `Find entity instance usedin`, `Find entity instance users` and `Find instance roles`.

6.7.1 getAssociatedModels

The getAssociatedModels method is a convenience method to retrieve the agglomerate domain of SdaiModels from an ASchemaInstance, see clause 4.6.

Possible error indicators:
SS_NOPN An SDAI session is not open.
FN_NAVL This function is not supported by this implementation.
SI_NEXS The schema instance does not exist.
SY_ERR An underlying system error occurred.

6.8 SdaiModel

The `SdaiModel` class shall represent the SDAI entity **sdai_model**, specified in ISO 10303-22:8.4.2 and the SDAI entity **sdai_model_contents**, specified in ISO 10303-22:8.4.3.

```
package SDAI.lang;
import SDAI.dictionary.*;

public class SdaiModel {
  public static final int NO_ACCESS;
  public static final int READ_ONLY;
  public static final int READ_WRITE;
  public String getName() throws SdaiException;
  public ESchema_definition getUnderlyingSchema() throws SdaiException;
  public String getUnderlyingSchemaString() throws SdaiException;
  public SdaiRepository getRepository() throws SdaiException;
  public boolean testChangeDate() throws SdaiException;
  public String getChangeDate() throws SdaiException;
  public long getChangeDateLong() throws SdaiException;
  public int getMode() throws SdaiException;
  public ASchemaInstance getAssociatedWith() throws SdaiException;
  public AEntity getInstances() throws SdaiException;
  public AEntityExtent getFolders() throws SdaiException;
  public AEntityExtent getPopulatedFolders() throws SdaiException;

  public void deleteSdaiModel() throws SdaiException;
  public void renameSdaiModel(String name) throws SdaiException;
  public void startReadOnlyAccess() throws SdaiException;
  public void promoteSdaiModelToRW() throws SdaiException;
  public void reduceSdaiModelToRO() throws SdaiException;
```

```
  public void endReadOnlyAccess() throws SdaiException;
  public void startReadWriteAccess() throws SdaiException;
  public void endReadWriteAccess() throws SdaiException;
  public EEntity_definition getEntityDefinition(String entity_name) throws SdaiException;

  //For late binding:
  public EEntity createEntityInstance(
    EEntity_definition entity_definition) throws SdaiException;
  //For early binding:
  public EEntity createEntityInstance(Class entity_definition) throws SdaiException;

  public EEntity copyInstance(EEntity source) throws SdaiException;
  public AEntity copyInstances(AEntity source) throws SdaiException;
  public void undoChanges() throws SdaiException;
  public void saveChanges() throws SdaiException;
  public int query(String where, EEntity entity, AEntity result) throws SdaiException;

  //For late binding:
  public AEntity getInstances(EEntity_definition entity_definition) throws SdaiException;
  public AEntity getInstances(EEntity_definition entity_definition[]) throws SdaiException;
  public AEntity getExactInstances(EEntity_definition entity_definition) throws SdaiException;
  //For early binding:
  public AEntity getInstances(Class entity_definition) throws SdaiException;
  public AEntity getInstances(Class entity_definition[]) throws SdaiException;
  public AEntity getExactInstances(Class entity_definition) throws SdaiException;

  public EEntity substituteInstance(EEntity old) throws SdaiException;
  //For late binding:
  public EEntity substituteInstance(EEntity old, EEntity_definition type) throws SdaiException;
  //For early binding:
  public EEntity substituteInstance(EEntity old, Class type) throws SdaiException;

  public ESchema_definition getDefinedSchema() throws SdaiException;
}
```

Note - Depending on the result of the ongoing ISO 10303-21 amendment standardization the following methods may be added in future to support section_language and section_context :

```
public A_string getContextIdentifiers() throws SdaiException;
public String getDefaultLanguage() throws SdaiException;
public void setDefaultLanguage(String value) throws SdaiException;
```

6.8.1 getMode

The `getMode` method returns an integer specifying the current access mode. The possible values are NO_ACCESS, READ_ONLY and READ_WRITE.

6.8.2 Delete SDAI-model

The `deleteSdaiModel` method shall implement the SDAI operation `Delete SDAI-model`, specified in ISO 10303-22:10.7.1.

6.8.3 Rename SDAI-model

The `renameSdaiModel` method shall implement the SDAI operation `Rename SDAI-model`, specified in ISO 10303-22:10.7.2.

6.8.4 Start read-only access

The `startReadOnlyAccess` method shall implement the SDAI operation `Start read-only access`, specified in ISO 10303-22:10.7.3.

6.8.5 Promote SDAI-model to read-write

The `promoteSdaiModelToRW` method shall implement the SDAI operation `Promote SDAI-model to read-write`, specified in ISO 10303-22:10.7.4.

6.8.6 Reduce SDAI-model to read-only

The `reduceSdaiModelToRO` method is an extension to the SDAI. It shall behave exactly like the end read-write access operation except that it changes the access mode of the SdaiModel from READ-WRITE to READ-ONLY.

NOTE - The transaction operations abort and commit resolve all pending changes.

Possible error indicators:
SS_NOPN An SDAI session is not open.
FN_NAVL This function is not supported by this implementation.
MO_NEXS The SDAI-model does not exist.
MX_NDEF The SDAI-model access is not defined.
MX_RO The SDAI-model access is read-only.
TR_RW The transaction is read-write and changes are unresolved.
TR_EAB The transaction ended abnormally.
SY_ERR An underlying system error occurred.

6.8.7 End read-only access

The `endReadOnlyAccess` method shall implement the SDAI operation `End read-only access`, specified in ISO 10303-22:10.7.5.

6.8.8 Start read-write access

The `startReadWriteAccess` method shall implement the SDAI operation `Start read-write access`, specified in ISO 10303-22:10.7.6.

6.8.9 End read-write access

The `endReadWriteAccess` method shall implement the SDAI operation `End read-write access`, specified in ISO 10303-22:10.7.7.

6.8.10 Get entity definition

The `getEntityDefinition` method shall implement the SDAI operation `Get entity definition`, specified in ISO 10303-22:10.7.8.

6.8.11 Create entity instance

The `createEntityInstance` methods shall implement the SDAI operation `Create entity instance`, specified in ISO 10303-22:10.7.9.

6.8.12 Copy application instance

The `copyInstance` method shall implement the SDAI operation `Copy application instance`, specified in ISO 10303-22:10.11.1.

6.8.13 copyInstances

The `copyInstances` method is a convenience method to copy entity instances from a provided aggregate to the model. The copies created are also collected into an aggregate which is returned by this method.

References between the source instances are copied such that they refer the new target instances. Otherwise this method work similarly as specified in the definition of the SDAI operation `Copy application instance`, see ISO 10303-22:10.11.1.

6.8.14 Undo changes

The `undoChanges` method shall implement the SDAI operation `Undo changes`, specified in ISO 10303-22:10.7.10.

6.8.15 Save changes

The `saveChanges` method shall implement the SDAI operation `Save changes`, specified in ISO 10303-22:10.7.11.

6.8.16 SDAI query

The `query` method shall implement the SDAI operation `SDAI query`, specified in ISO 10303-22:10.4.14 in the case that the `query_source` is an `SDAI-model`.

6.8.17 Get instances

The `getInstances` methods shall return a read-only aggregate of instances. Several overloaded methods are specified to retrieve either all instances or only instances for one or several entity data types and for late and early binding access.

The returned aggregate shall be of type AEntity. Applications can cast this aggregate to an aggregate of the specific entity.

Possible error indicators:
SS_NOPN An SDAI session is not open.
MO_NEXS The SDAI-model does not exist.
MX_NDEF The SDAI-model access is not defined.
ED_NDEF Entity definition is not defined.
ED_NVLD Entity definition invalid.
VA_NVLD Value invalid.
SY_ERR An underlying system error occurred.

6.8.18 Get exact instances

The `getExactInstances` methods shall return a read-only aggregate containing all the instances of a complex entity data type specified by the parameter `entity_definition`. The variation of the method for early binding shall have a parameter of type Java Class whose allowed value is an entity class, see subclause 8.9.

The returned aggregate shall be of type AEntity. Applications can cast this aggregate to an aggregate of the specific entity.

Possible error indicators:
SS_NOPN An SDAI session is not open.
MO_NEXS The SDAI-model does not exist.
MX_NDEF The SDAI-model access is not defined.
ED_NDEF Entity definition is not defined.
ED_NVLD Entity definition invalid.
VA_NVLD Value invalid.
SY_ERR An underlying system error occurred.

6.8.19 substituteInstance

The convenience methods `substituteInstance` move an entity instances to another model and/or to change the type of the instance. This method returns either the same Java object for the entity instances or returns a new Java object for the same entity instance and mark the old Java object to be invalid. The persistent label is not changed by this method as long as the entity instance remains in the same SdaiRepository. References from other entity instances within SdaiModels in READ-WRITE access mode are updated to refer the new object.

If the type of the instance is changed, the attribute values of all single entity data types, which exist in both, the old and the new complex entity data type, shall be maintained. The attributes of new entity data types shall be unset.

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.
ED_NDEF The entity definition is not defined.
EI_NVLD The entity definition invalid.
EI_NVLD The entity instance invalid.
SY_ERR An underlying system error occurred.

6.8.20 Get defined schema

The `getDefinedSchema` method is a convenience method that shall find the `schema_definition` instance in the case that the `sdai_model` manages dictionary data. Otherwise, the null value shall be returned.

NOTE - The same instance of schema_definition can be found by traversing to:

```
sdai_model.contents -> sdai_model_contents
sdai_model_contents.folders[i] -> entity_extent
entity_extent.instances[i] -> EEntity
EEntity => ESchema_definition
```

Possible error indicators:
SS_NOPN An SDAI session is not open.
MO_NEXS The SDAI-model does not exist.
MX_NDEF The SDAI-model access is not defined.
FN_NAVL This function is not supported by this implementation.
SY_ERR An underlying system error occurred.

6.9 ASdaiModel

The `ASdaiModel` class shall represent an aggregate of the SDAI entity **sdai_model**, specified in ISO 10303-22:8.4.2.

```
package SDAI.lang;

public class ASdaiModel implements Aggregate {
  public boolean isMember(SdaiModel value) throws SdaiException;
  public SdaiModel getByIndex(int index) throws SdaiException;
  public SdaiModel getCurrentMember(SdaiIterator iter) throws SdaiException;
}
```

6.10 EntityExtent

The `EntityExtent` class shall represent the SDAI entity **entity_extent**, specified in ISO 10303-22:8.4.4.

```
package SDAI.lang;
import SDAI.dictionary.*;

public class EntityExtent {
  EEntity_definition getDefinition() throws SdaiException;
  String getDefinitionString() throws SdaiException;
  AEntity getInstances() throws SdaiException;
  SdaiModel getOwnedBy()throws SdaiException;
}
```

For implementations supporting early binding it shall be possible to cast the aggregate returned by `getInstances` to an early binding aggregate of the corresponding entity definition.

6.11 AEntityExtent

The `AEntityExtent` class shall implement an aggregate of the SDAI entity **entity_extent**, specified in ISO 10303-22:8.4.4.

```
package SDAI.lang;

public class AEntityExtent implements Aggregate {
  public boolean isMember(EntityExtent value) throws SdaiException;
  public EntityExtent getCurrentMember(SdaiIterator iter) throws SdaiException;
}
```

6.12 LOGICAL

The Java class ELogical supports the transformation between the EXPRESS LOGICAL data type and the Java primitive data type int.

```
package SDAI.lang;

public final class ELogical {

    public static final int FALSE = 1;
    public static final int TRUE = 2;
    public static final int UNKNOWN = 3;

    public static final String values[] = {"FALSE", "TRUE", "UNKNOWN"};

    private ELogical() {
    }

    public static int toInt(String str) {
        for (int i = 0; i < 3; i++) {
            if (values[i].equalsIgnoreCase(str)) return i + 1;
        }
        return 0;
    }

    public static String toString(int value) {
        if (value == 0) return "unset";
        return values[value - 1];
    }
}
```

7 Late binding

This clause defines the Java interfaces and classes for late binding of the **SDAI_parameter_data_schema** as specified in clause 9 of ISO 10303-22. Late binding support for application schemas is mandatory.

7.1 Late binding base types

Late binding attribute access is specified in ISO 10303-22:4.8. For late binding attribute access the actual Java type of an attribute-value or aggregate-member is either int, double, boolean or Object. The return value of the `testAttribute`, `testCurrentMember` or `testByIndex` methods determines the actual Java type.

The possible return values are as follows:

— 0 for an unset value;

— 1 for the Java type `Object` ;

— 2 for the Java type `int`;

— 3 for the Java type `double`;

— 4 for the Java type `boolean`.

Applications may use this return value to select the proper late binding method to retrieve an attribute or aggregate member value of type `Object`, `int`, `double` or `boolean`.

7.2 Select data types for late binding

A Java array of EDefined_type, called the **select-array**, is used to distinguish between EXPRESS SELECT values for attributes and members of aggregates. The following late binding access methods have a `select` parameter for this select-array:

— Test attribute;

— Put attribute;

— Test by index;

— Put by index;

— Test current member;

— Put current member;

— Add before current member;

— Add after current member;

— Add by index;

— Add unordered;

— Is member;

— Remove unordered;

— Create aggregate instance;

— Create aggregate instance by index;

— Create aggregate instance as current member;

— Create aggregate instance before current member;

— Create aggregate instance after current member;

— Add aggregate instance by index;

— Create aggregate instance unordered.

NOTE - The select-array corresponds to the attribute data_type of the possible EXPRESS specification given in example 11 of ISO 10303-22:9.4.8.

The order of the defined data types in this array shall follow the order of KEYWORDS as defined in part-21, and in the technical corrigendum to ISO 10303-21:11.1.8.

For the SDAI methods t`estAttribute`, `testCurrentMember` and t`estByIndex` an implementation of this language binding shall fill the select-array with the select information and set all unused array elements to null. The application is responsible for supplying a sufficiently big array. In the case when the size of the array is too small to store the select information, an SdaiException VA_NVLD shall be thrown.

To use all other methods listed above, an application should fill the select-array with proper values. In the case when the attribute or aggregation type is not of the select type, an application can pass the `null` value for the `select` parameter. If the values in the select-array do not fit with the underlying express schema, the SdaiException VA_NVLD shall be thrown.

NOTE - For an example of a late binding application dealing with SELECTs, see D.3.

7.3 EEntity

The `EEntity` interface shall represent the SDAI entities **application_instance**, specified in ISO 10303-22:9.4.3 and **dictionary_instance**, specified in ISO 10303-22:9.4.5.

```
public interface EEntity {
  Object get_object(EAttribute attribute) throws SdaiException;
  int get_int(EAttribute attribute) throws SdaiException;
  double get_double(EAttribute attribute) throws SdaiException;
  boolean get_boolean(EAttribute attribute) throws SdaiException;
  AEntity get_inverse(EInverse_attribute attribute, ASdaiModel domain) throws SdaiException;
  int testAttribute(EAttribute attribute, EDefined_type[] select) throws SdaiException;
  SdaiModel findEntityInstanceSdaiModel() throws SdaiException;
  //For late binding:
  EEntity_definition getInstanceType() throws SdaiException;
```

```
  //For late binding:
  boolean isInstanceOf(EEntity_definition type) throws SdaiException;
  //For late binding:
  boolean isKindOf(EEntity_definition type) throws SdaiException;
  void findEntityInstanceUsers(ASdaiModel domain, AEntity result) throws SdaiException;
  void findEntityInstanceUsedin(EAttribute role, ASdaiModel domain, AEntity result)
    throws SdaiException;
  int getAttributeValueBound(EAttribute attribute) throws SdaiException;
  void findInstanceRoles(ASdaiModel domain, AAttribute result) throws SdaiException;
  void findInstanceDataTypes(ANamed_type result) throws SdaiException;
  void deleteApplicationInstance() throws SdaiException;
  void set(EAttribute attribute, Object value, EDefined_type select[]) throws SdaiException;
  void set(EAttribute attribute, int value, EDefined_type select[]) throws SdaiException;
  void set(EAttribute attribute, double value, EDefined_type select[]) throws SdaiException;
  void unsetAttributeValue(EAttribute Attribute) throws SdaiException;
  Aggregate createAggregate(EAttribute Attribute, EDefined_type select[]) throws SdaiException;
  String getPersistentLabel() throws SdaiException;
  String getDescription() throws SdaiException;
  int validateWhereRule(EWhere_rule Rule) throws SdaiException;
  boolean validateRequiredExplicitAttributesAssigned(AAttribute NonConf) throws SdaiException;
  boolean validateInverseAttributes(AAttribute NonConf) throws SdaiException;
  int validateExplicitAttributesReferences(AAttribute NonConf) throws SdaiException;
  int validateAggregatesSize(AAttribute NonConf) throws SdaiException;
  boolean validateAggregatesUniqueness(AAttribute NonConf) throws SdaiException;
  boolean validateArrayNotOptional(AAttribute NonConf) throws SdaiException;
  int validateStringWidth(AAttribute NonConf) throws SdaiException;
  int validateBinaryWidth(AAttribute NonConf) throws SdaiException;
  int validateRealPrecision(AAttribute NonConf) throws SdaiException;
  boolean compareValueBoolean(EEntity instance) throws SdaiException;
  int compareValueLogical(EEntity instance) throws SdaiException;
  String toString();
}
```

7.3.1 Get attribute

The `get_object`, `get_int`, `get_boolean` and `get_double` methods shall implement the SDAI operation `Get attribute`, specified in ISO 10303-22:10.10.1.

In addition to error indicators defined in ISO 10303-22, these methods shall also use the following error indicator: VT_NVLD, The value type is invalid.

7.3.2 Get inverse attribute

The `get_inverse` method shall implement the SDAI operation `Get attribute`, specified in ISO 10303-22:10.10.1 for late binding access of inverse attributes. The `domain` parameter specifies the agglomerate domain as defined in clause 4.6.

7.3.3 Test attribute

The `testAttribute` method shall implement the SDAI operation `Test attribute`, specified in ISO 10303-22:10.10.2. See clause 7.1 for the return value.

See clause 7.2 for the `select` parameter.

7.3.4 Find entity instance SDAI-model

The `findEntityInstanceSdaiModel` method shall implement the SDAI operation `Find entity instance SDAI-model`, specified in ISO 10303-22:10.10.3.

7.3.5 Get instance type

The `getInstanceType` method shall implement the SDAI operation `Get instance type`, specified in ISO 10303-22:10.10.4.

7.3.6 Is instance of

The `isInstanceOf` method shall implement the SDAI operation `Is instance of`, specified in ISO 10303-22:10.10.5.

Note: No early binding method for this operation is given because applications can directly compare the Java classes, using the Java method `getClass`.

7.3.7 Is kind of

The `isKindOf` method shall implement the SDAI operation `Is kind of`, specified in ISO 10303-22:10.10.6.

Note: No early binding method for this operation is given because applications can directly use the Java operator `instanceOf` or the Java `Class` method is`Instance` to compare the entity instance with an entity interface.

7.3.8 Find entity instance users

The `findEntityInstanceUsers` method shall implement the SDAI operation `Find entity instance users`, specified in ISO 10303-22:10.10.8.

In the case when the parameter `domain` is null, the domain includes only the SdaiModel to which this instance belongs.

7.3.9 Find entity instance usedin

The `findEntityInstanceUsedin` method shall implement the SDAI operation `Find entity instance usedin`, specified in ISO 10303-22:10.10.9.

In the case when the parameter `domain` is null, the domain includes only the SdaiModel to which this instance belongs.

7.3.10 Get attribute value bound

The `getAttributeValueBound` method shall implement the SDAI operation `Get attribute value bound`, specified in ISO 10303-22:10.10.10.

7.3.11 Find instance roles

The `findInstanceRoles` method shall implement the SDAI operation `Find instance roles`, specified in ISO 10303-22:10.10.11.

In the case when the parameter `domain` is null, the domain includes only the SdaiModel to which this instance belongs.

7.3.12 Find instance data types

The `findInstanceDataTypes` method shall implement the SDAI operation `Find instance data types`, specified in ISO 10303-22:10.10.12.

7.3.13 Delete application instance

The `deleteApplicationInstance` method shall implement the SDAI operation `Delete application instance`, specified in ISO 10303-22:10.11.2.

7.3.14 Put attribute

The `set` methods shall implement the SDAI operation `Put attribute`, specified in ISO 10303-22:10.11.3.

See clause 7.2 for the `select` parameter.

7.3.15 Unset attribute value

The `unsetAttributeValue` method shall implement the SDAI operation `Unset attribute value`, specified in ISO 10303-22:10.11.4.

7.3.16 Create aggregate instance

The `createAggregate` method shall implement the SDAI operation `Create aggregate instance`, specified in ISO 10303-22:10.11.5.

See clause 7.2 for the `select` parameter.

7.3.17 Get persistent label

The `getPersistentLabel` method shall implement the SDAI operation `Get persistent label`, specified in ISO 10303-22:10.11.6.

7.3.18 Get description

The `getDescription` method shall implement the SDAI operation `Get description`, specified in ISO 10303-22:10.11.8.

7.3.19 Validate where rule

The `validateWhereRule` method shall implement the SDAI operation `Validate where rule`, specified in ISO 10303-22:10.11.9.

7.3.20 Validate required explicit attributes assigned

The validateRequiredExplicitAttributesAssigned method shall implement the SDAI operation Validate required explicit attributes assigned, specified in ISO 10303-22:10.11.10.

7.3.21 Validate inverse attributes

The `validateInverseAttributes` method shall implement the SDAI operation `Validate inverse attributes`, specified in ISO 10303-22:10.11.11.

7.3.22 Validate explicit attributes references

The `validateExplicitAttributesReferences` method shall implement the SDAI operation `Validate explicit attributes references`, specified in ISO 10303-22:10.11.12.

7.3.23 Validate aggregates size

The `validateAggregatesSize` method shall implement the SDAI operation `Validate aggregates size`, specified in ISO 10303-22:10.11.13.

7.3.24 Validate aggregates uniqueness

The `validateAggregatesUniqueness` method shall implement the SDAI operation `Validate aggregates uniqueness`, specified in ISO 10303-22:10.11.14.

7.3.25 Validate array not optional

The `validateArrayNotOptional` method shall implement the SDAI operation `Validate array not optional`, specified in ISO 10303-22:10.11.15.

7.3.26 Validate string width

The `validateStringWidth` method shall implement the SDAI operation `Validate string width`, specified in ISO 10303-22:10.11.16.

7.3.27 Validate binary width

The `validateBinaryWidth` method shall implement the SDAI operation `Validate binary width`, specified in ISO 10303-22:10.11.17.

7.3.28 Validate real precision

The `validateRealPrecision` method shall implement the SDAI operation `Validate real precision`, specified in ISO 10303-22:10.11.18.

7.3.29 compareValueBoolean

The `compareValueBoolean` method is a convenience method to compare the complex entity data type and all attribute values of two entity instances. The method shall return true if the type of the instances and the values of all attributes are identical. Otherwise, the method shall return false.

Possible error indicators:
SS_NOPN An SDAI session is not open.
EI_NEXS Entity instance does not exist.
FN_NAVL This function is not supported by this implementation.
SY_ERR An underlying system error occurred.

7.3.30 compareValueLogical

The `compareValueLogical` method is a convenience method for a **value comparison**, specified in ISO 10303-11:12.2.1, of two entity instances. The return value shall represent the LOGICAL value of the value comparison.

Possible error indicators:
SS_NOPN An SDAI session is not open.
EI_NEXS Entity instance does not exist.
FN_NAVL This function is not supported by this implementation.
SY_ERR An underlying system error occurred.

7.3.31 toString

The `toString` method is a convenience method useful, for example, for debugging. It shall return a Java String which contains the persistent identifier and the values of the attributes of this instance.

Note - Implementations may choose a format similar to the entity instances mapping in ISO 10303-21.

Possible error indicators:
SS_NOPN An SDAI session is not open.
EI_NEXS Entity instance does not exist.
FN_NAVL This function is not supported by this implementation.
SY_ERR An underlying system error occurred.

7.4 Aggregate

The SDAI entity **aggregate_instance**, specified in ISO 10303-22:9.4.11, its subtypes, and the operations defined on it shall be represented by the Java interface Aggregate.

```
public interface Aggregate
{
  SDAI.dictionary.EAggregation_type getAggregationType()
```

```
    throws SdaiException;
  int getMemberCount() throws SdaiException;
  boolean isMember(Object value, EDefined_type select[]) throws SdaiException;
  boolean isMember(int value, EDefined_type select[]) throws SdaiException;
  boolean isMember(double value, EDefined_type select[]) throws SdaiException;
  boolean isMember(boolean value, EDefined_type select[]) throws SdaiException;
  SdaiIterator createIterator() throws SdaiException;
  void attachIterator(SdaiIterator iter) throws SdaiException;
  int getLowerBound() throws SdaiException;
  int getUpperBound() throws SdaiException;
  int query(String where, EEntity entity, AEntity result) throws SdaiException;
  Object getByIndexObject(int index) throws SdaiException;
  int getByIndexInt(int index) throws SdaiException;
  double getByIndexDouble(int index) throws SdaiException;
  boolean getByIndexBoolean(int index) throws SdaiException;
  int getValueBoundByIndex(int index) throws SdaiException;
  void setByIndex(int index, Object value, EDefined_type select[]) throws SdaiException;
  void setByIndex(int index, int value, EDefined_type select[]) throws SdaiException;
  void setByIndex(int index, double value, EDefined_type select[]) throws SdaiException;
  void setByIndex(int index, boolean value, EDefined_type select[]) throws SdaiException;
  Aggregate createAggregateByIndex(int index, EDefined_type select[]) throws SdaiException;
  int testByIndex(int index, EDefined_type select[]) throws SdaiException;
  int testByIndex(int index) throws SdaiException;
  int getLowerIndex() throws SdaiException;
  int getUpperIndex() throws SdaiException;
  void unsetValueByIndex(int index) throws SdaiException;
  void reindexArray() throws SdaiException;
  void resetArrayIndex(int lower, int upper) throws SdaiException;
  void addByIndex(int index, Object value, EDefined_type select[]) throws SdaiException;
  void addByIndex(int index, int value, EDefined_type select[]) throws SdaiException;
  void addByIndex(int index, double value, EDefined_type select[]) throws SdaiException;
  void addByIndex(int index, boolean value, EDefined_type select[]) throws SdaiException;
  Aggregate addAggregateByIndex(int index, EDefined_type select[]) throws SdaiException;
  void removeByIndex(int index) throws SdaiException;
  void deleteNonPersistentList() throws SdaiException;
  void addUnordered(Object value, EDefined_type select[]) throws SdaiException;
  void addUnordered(int value, EDefined_type select[]) throws SdaiException;
  void addUnordered(double value, EDefined_type select[]) throws SdaiException;
  void addUnordered(boolean value, EDefined_type select[]) throws SdaiException;
  Aggregate createAggregateUnordered(EDefined_type select[]) throws SdaiException;
  void removeUnordered(Object value, EDefined_type select[]) throws SdaiException;
  void removeUnordered(int value, EDefined_type select[]) throws SdaiException;
  void removeUnordered(double value, EDefined_type select[]) throws SdaiException;
  void removeUnordered(boolean value, EDefined_type select[]) throws SdaiException;
  int testCurrentMember(SdaiIterator iter, EDefined_type select[]) throws SdaiException;
  int testCurrentMember(SdaiIterator iter) throws SdaiException;
  Object getCurrentMemberObject(SdaiIterator iter) throws SdaiException;
  int getCurrentMemberInt(SdaiIterator iter) throws SdaiException;
  double getCurrentMemberDouble(SdaiIterator iter) throws SdaiException;
  boolean getCurrentMemberBoolean(SdaiIterator iter) throws SdaiException;
  Aggregate createAggregateAsCurrentMember(SdaiIterator iter, EDefined_type select[])
    throws SdaiException;
  void setCurrentMember(SdaiIterator iter, Object value, EDefined_type select[])
```

```
    throws SdaiException;
  void setCurrentMember(SdaiIterator iter, int value, EDefined_type select[]) throws SdaiException;
  void setCurrentMember(SdaiIterator iter, double value, EDefined_type select[])
    throws SdaiException;
  void setCurrentMember(SdaiIterator iter, boolean value, EDefined_type select[])
    throws SdaiException;
  void addBefore(SdaiIterator iter, Object value, EDefined_type select[]) throws SdaiException;
  void addBefore(SdaiIterator iter, int value, EDefined_type select[]) throws SdaiException;
  void addBefore(SdaiIterator iter, double value, EDefined_type select[]) throws SdaiException;
  void addBefore(SdaiIterator iter, boolean value, EDefined_type select[]) throws SdaiException;
  void addAfter(SdaiIterator iter, Object value, EDefined_type select[]) throws SdaiException;
  void addAfter(SdaiIterator iter, int value, EDefined_type select[]) throws SdaiException;
  void addAfter(SdaiIterator iter, double value, EDefined_type select[]) throws SdaiException;
  void addAfter(SdaiIterator iter, boolean value, EDefined_type select[]) throws SdaiException;
  Aggregate createAggregateBefore(SdaiIterator iter, EDefined_type select[]) throws SdaiException;
  Aggregate createAggregateAfter(SdaiIterator iter, EDefined_type select[]) throws SdaiException;
  void clear() throws SdaiException;
  void String toString();
}
```

7.4.1 Get aggregation type

The `getAggregationType` method is a convenience method to retrieve information from the SDAI_dictionary_schema on the particular type.

Possible error indicators:
AI_NEXS The aggregate instance does not exist.
FN_NAVL This function is not supported by this implementation.
SY_ERR An underlying system error occurred.

7.4.2 Get member count

The `getMemberCount` method shall implement the SDAI operation `Get member count`, specified in ISO 10303-22:10.12.1.

7.4.3 Is member

The `isMember` method shall implement the SDAI operation `Is member`, specified in ISO 10303-22:10.12.2.

See clause 7.2 for the `select` parameter.

7.4.4 Create iterator

The `createIterator` method shall implement the SDAI operation `Create iterator`, specified in ISO 10303-22:10.12.3.

7.4.5 Attach iterator

The `attachIterator` method is a convenience method which shall support the reuse of an existing SdaiIterator object, created with `Create iterator`, for another aggregate. The SDAI operation `Create iterator` is, specified in ISO 10303-22:10.12.3.

7.4.6 Get lower bound

The `getLowerBound` method shall implement the SDAI operation `Get lower bound`, specified in ISO 10303-22:10.12.9.

7.4.7 Get upper bound

The `getUpperBound` method shall implement the SDAI operation `Get upper bound`, specified in ISO 10303-22:10.12.10.

7.4.8 SDAI query

The `query` method shall implement the SDAI operation `SDAI query`, specified in ISO 10303-22:10.4.14, in the case when the `query source` is an aggregate.

7.4.9 Get by index

The `getByIndexObject`, `getByIndexInt`, `getByIndexBoolean` and `getByIndexDouble` methods shall implement the SDAI operation `Get by index`, specified in ISO 10303-22:10.15.1. Further overloaded methods shall be defined in classes implementing this interface.

In addition to error indicators defined in ISO 10303-22, these methods shall also use the following error indicator: VT_NVLD, The value type is invalid.

7.4.10 Get value bound by index

The `getValueBoundByIndex` method shall implement the SDAI operation `Get value bound by index`, specified in ISO 10303-22:10.15.4.

7.4.11 Put by index

The `setByIndex` methods shall implement the SDAI operation `Put by index`, specified in ISO 10303-22:10.16.1. Further overloaded methods shall be defined in classes implementing this interface.

See clause 7.2 for the `select` parameter.

7.4.12 Create aggregate instance by index

The `createAggregateByIndex` methods shall implement the SDAI operation `Create aggregate instance by index`, specified in ISO 10303-22:10.16.2.

See clause 7.2 for the `select` parameter.

7.4.13 Test by index for late binding

The `testByIndex` method shall implement the SDAI operation `Test by index`, specified in ISO 10303-22:10.17.1. See clause 7.1 for the return value

NOTE - The return value may be used to select the proper method implementing `Get Attribute` operation.

See clause 7.2 for the `select` parameter.

7.4.14 Get lower index

The `getLowerIndex` method shall implement the SDAI operation `Get lower index`, specified in ISO 10303-22:10.17.3.

7.4.15 Get upper index

The `getUpperIndex` method shall implement the SDAI operation `Get upper index`, specified in ISO 10303-22:10.17.4.

7.4.16 Unset value by index

The `unsetValueByIndex` method shall implement the SDAI operation `Unset value by index`, specified in ISO 10303-22:10.18.1.

7.4.17 Reindex array

The `reindexArray` method shall implement the SDAI operation `Reindex array`, specified in ISO 10303-22:10.18.3.

7.4.18 Reset array index

The `resetArrayIndex` method shall implement the SDAI operation `Reset array index`, specified in ISO 10303-22:10.18.4.

7.4.19 Add by index

The `addByIndex` methods shall implement the SDAI operation `Add by index`, specified in ISO 10303-22:10.19.3. Further overloaded methods shall be defined in classes implementing this interface.

See clause 7.2 for the `select` parameter.

7.4.20 Add aggregate instance by index

The `addAggregateByIndex` method shall implement the SDAI operation `Add aggregate instance by index`, specified in ISO 10303-22:10.19.6.

See clause 7.2 for the `select` parameter.

7.4.21 Remove by index

The `removeByIndex` method shall implement the SDAI operation `Remove by index`, specified in ISO 10303-22:10.19.7.

7.4.22 Add unordered

The `addUnordered` methods shall implement the SDAI operation `Add unordered`, specified in ISO 10303-22:10.14.1.

See clause 7.2 for the `select` parameter.

7.4.23 Create aggregate instance unordered

The `createAggregateUnordered` method shall implement the SDAI operation `Create aggregate instance unordered`, specified in ISO 10303-22:10.14.2.

See clause 7.2 for the `select` parameter.

7.4.24 Remove unordered

The `removeUnordered` methods shall implement the SDAI operation `Remove unordered`, specified in ISO 10303-22:10.14.3.

See clause 7.2 for the `select` parameter.

7.4.25 Test current member

The `testCurrentMember` methods shall implement the SDAI operation `Test current member`, specified in ISO 10303-22:10.17.2. See clause 7.1 for the return value.

NOTE - The return value may be used to select the proper method implementing `Get Attribute` operation.

See clause 7.2 for the `select` parameter.

7.4.26 Get current member

The `getCurrentMemberObject`, `getCurrentMemberInt` and `getCurrentMemberDouble` methods shall implement the SDAI operation `Get current member`, specified in ISO 10303-22:10.12.7.

In addition to error indicators defined in ISO 10303-22, these methods shall also use the following error indicator: VT_NVLD, The value type is invalid.

7.4.27 Create aggregate instance as current member

The `createAggregateAsCurrentMember` method shall implement the SDAI operation `Create aggregate instance as current member`, specified in ISO 10303-22:10.13.1.

See clause 7.2 for the `select` parameter.

7.4.28 Put current member

The `setCurrentMember` methods shall implement the SDAI operation `Put current member`, specified in ISO 10303-22:10.13.2.

See clause 7.2 for the `select` parameter.

7.4.29 Add before current member

The `addBefore` methods shall implement the SDAI operation `Add before current member`, specified in ISO 10303-22:10.19.1.

See clause 7.2 for the `select` parameter.

7.4.30 Add after current member

The `addAfter` methods shall implement the SDAI operation `Add after current member`, specified in ISO 10303-22:10.19.2.

See clause 7.2 for the `select` parameter.

7.4.31 Create aggregate instance before current member

The `createAggregateBefore` method shall implement the SDAI operation `Create aggregate instance before current member`, specified in ISO 10303-22:10.19.4.

See clause 7.2 for the `select` parameter.

7.4.32 Create aggregate instance after current member

The `createAggregateAfter` method shall implement the SDAI operation `Create aggregate instance after current member`, specified in ISO 10303-22:10.19.5.

See clause 7.2 for the `select` parameter.

7.4.33 Clear

The `clear` method is a convenience method which shall be used to clear an aggregate. After successful invocation of this method, the aggregate and any nested aggregates shall be in the same state as after the “create” aggregate operations.

Possible error indicators:

TR_NRW The transaction is not read-write.
TR_NAVL The transaction is currently not available.
MX_NRW The SDAI-model access is not read-write.
AI_NEXS The aggregate instance 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.

7.4.34 toString

The `toString` method is a convenience method useful, for example, for debugging. It shall return a Java String which contains the values of the members of this aggregate.

Note - Implementations may choose a format similar to the aggregate mapping in part-21.

7.5 SdaiIterator

The `SdaiIterator` class shall represent the SDAI mechanism `iterator` defined in ISO 10303-22:3.3.9.

```
public class SdaiIterator {
  void beginning() throws SdaiException;
  boolean next() throws SdaiException;
  int getValueBound() throws SdaiException;
  void remove() throws SdaiException;
  void end() throws SdaiException;
  boolean previous() throws SdaiException;
  void unset() throws SdaiException;
}
```

Note - The SDAI operation `Delete iterator`, specified in ISO 10303-22:10.12.4, is implicitly realized by the Java garbage collection.

7.5.1 Beginning

The `beginning` method shall implement the SDAI operation `Beginning`, specified in ISO 10303-22:10.12.5.

7.5.2 Next

The `next` method shall implement the SDAI operation `Next`, specified in ISO 10303-22:10.12.6.

7.5.3 Get value bound by iterator

The `getValueBound` method shall implement the SDAI operation `Get value bound by iterator`, specified in ISO 10303-22:10.12.8.

7.5.4 Remove current member

The `remove` method shall implement the SDAI operation `Remove current member`, specified in ISO 10303-22:10.13.3.

7.5.5 End

The `end` method shall implement the SDAI operation `end`, specified in ISO 10303-22:10.15.2.

7.5.6 Previous

The `previous` method shall implement the SDAI operation `Previous`, specified in ISO 10303-22:10.15.3.

7.5.7 Unset value current member

The `unset` method shall implement the SDAI operation `Unset value current member`, specified in ISO 10303-22:10.18.2.

7.6 Aggregates of basic types

Simple aggregates and aggregates of aggregates for the Java types `int`, `double`, `boolean`, `String` and `Binary` and for enumerations shall be defined for early binding access. It is up to implementation to provide aggregates of higher order as needed. Alternatively, applications may use the late binding access methods as defined in interface `Aggregate`. Furthermore, general purpose non-nested aggregates for `EEntity` and `AEntity` for early binding access shall be defined.

7.6.1 AEntity

The `AEntity` class shall be used to simplify the creation of early binding aggregates of subtypes of application entities which have no SELECT-valued attributes. While extending it the methods `getByIndex` and `getCurrentMember` shall be added. These methods shall return the actual Java-interface associated with the type of the application entity for which these aggregates are constructed.

```
public class AEntity implements Aggregate
{
  public boolean isMember(EEntity value) throws SdaiException;
  public EEntity getByIndexEntity(int index) throws SdaiException;
  public void setByIndex(int index, EEntity value) throws SdaiException;
  public void addByIndex(int index, EEntity value) throws SdaiException;
  public void addUnordered(EEntity value) throws SdaiException;
  public void removeUnordered(EEntity value) throws SdaiException;
  public EEntity getCurrentMemberEntity(SdaiIterator iter) throws SdaiException;
  public void setCurrentMember(SdaiIterator iter, EEntity value) throws SdaiException;
  public void addBefore(SdaiIterator iter, EEntity value) throws SdaiException;
  public void addAfter(SdaiIterator iter, EEntity value) throws SdaiException;
}
```

7.6.2 A_string

The A_string class shall represent aggregates of Java type String.

```
public class A_string implements Aggregate
{
  public boolean isMember(String value) throws SdaiException;
  public String getByIndex(int index) throws SdaiException;
  public void setByIndex(int index, String value) throws SdaiException;
  public void addByIndex(int index, String value) throws SdaiException;
  public void addUnordered(String value) throws SdaiException;
  public void removeUnordered(String value) throws SdaiException;
  public String getCurrentMember(SdaiIterator iter) throws SdaiException;
  public void setCurrentMember(SdaiIterator iter, String value) throws SdaiException;
  public void addBefore(SdaiIterator iter, String value) throws SdaiException;
  public void addAfter(SdaiIterator iter, String value) throws SdaiException;
}
```

7.6.3 Aa_string

The Aa_string class shall represent aggregates of aggregates of Java type String.

```
public class Aa_string implements Aggregate
{
  public boolean isMember(A_string value) throws SdaiException;
  public A_string getByIndex(int index) throws SdaiException;
  public A_string createAggregateByIndex(int index) throws SdaiException;
  public A_string addAggregateByIndex(int index) throws SdaiException;
  public A_string createAggregateUnordered() throws SdaiException;
  public A_string getCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_string createAggregateAsCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_string createAggregateBefore(SdaiIterator iter) throws SdaiException;
  public A_string createAggregateAfter(SdaiIterator iter) throws SdaiException;
  public void removeUnordered(A_string value) throws SdaiException;
}
```

7.6.4 A_double

The A_double class shall represent aggregates of Java type double.

```
public class A_double implements Aggregate
{
  public boolean isMember(double value) throws SdaiException;
  public double getByIndex(int index) throws SdaiException;
  public void setByIndex(int index, double value) throws SdaiException;
  public void addByIndex(int index, double value) throws SdaiException;
  public void addUnordered(double value) throws SdaiException;
  public void removeUnordered(double value) throws SdaiException;
  public double getCurrentMember(SdaiIterator iter) throws SdaiException;
  public void setCurrentMember(SdaiIterator iter, double value) throws SdaiException;
  public void addBefore(SdaiIterator iter, double value) throws SdaiException;
  public void addAfter(SdaiIterator iter, double value) throws SdaiException;
}
```

7.6.5 Aa_double

The Aa_double class shall represent aggregates of aggregates of Java type double.

```
public class Aa_double implements Aggregate
{
  public boolean isMember(A_double value) throws SdaiException;
  public A_double getByIndex(int index) throws SdaiException;
  public A_double createAggregateByIndex(int index) throws SdaiException;
  public A_double addAggregateByIndex(int index) throws SdaiException;
  public A_double createAggregateUnordered() throws SdaiException;
  public A_double getCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_double createAggregateAsCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_double createAggregateBefore(SdaiIterator iter) throws SdaiException;
  public A_double createAggregateAfter(SdaiIterator iter) throws SdaiException;
  public void removeUnordered(A_double value) throws SdaiException;
}
```

7.6.6 A_integer

The A_integer class shall represent aggregates of Java type integer.

```
public class A_integer implements Aggregate
{
  public boolean isMember(int value) throws SdaiException;
  public int getByIndex(int index) throws SdaiException;
  public void setByIndex(int index, int value) throws SdaiException;
  public void addByIndex(int index, int value) throws SdaiException;
  public void addUnordered(int value) throws SdaiException;
  public void removeUnordered(int value) throws SdaiException;
  public int getCurrentMember(SdaiIterator iter) throws SdaiException;
  public void setCurrentMember(SdaiIterator iter, int value) throws SdaiException;
  public void addBefore(SdaiIterator iter, int value) throws SdaiException;
  public void addAfter(SdaiIterator iter, int value) throws SdaiException;
}
```

7.6.7 Aa_integer

The Aa_integer class shall represent aggregates of aggregates of Java type integer.

```
public class Aa_integer implements Aggregate
{
  public boolean isMember(A_integer value) throws SdaiException;
  public A_integer getByIndex(int index) throws SdaiException;
  public A_integer createAggregateByIndex(int index) throws SdaiException;
  public A_integer addAggregateByIndex(int index) throws SdaiException;
  public A_integer createAggregateUnordered() throws SdaiException;
  public A_integer getCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_integer createAggregateAsCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_integer createAggregateBefore(SdaiIterator iter) throws SdaiException;
  public A_integer createAggregateAfter(SdaiIterator iter) throws SdaiException;
```

```
  public void removeUnordered(A_integer value) throws SdaiException;
}
```

7.6.8 A_enumeration

The A_enumeration class shall represent aggregates of enumerations.

```
public class A_enumeration implements Aggregate
{
  public boolean isMember(int value) throws SdaiException;
  public int getByIndex(int index) throws SdaiException;
  public void setByIndex(int index, int value) throws SdaiException;
  public void addByIndex(int index, int value) throws SdaiException;
  public void addUnordered(int value) throws SdaiException;
  public void removeUnordered(int value) throws SdaiException;
  public int getCurrentMember(SdaiIterator iter) throws SdaiException;
  public void setCurrentMember(SdaiIterator iter, int value) throws SdaiException;
  public void addBefore(SdaiIterator iter, int value) throws SdaiException;
  public void addAfter(SdaiIterator iter, int value) throws SdaiException;
}
```

7.6.9 Aa_ enumeration

The Aa_enumeration class shall represent aggregates of aggregates of enumerations.

```
public class Aa_enumeration implements Aggregate
{
  public boolean isMember(A_enumeration value) throws SdaiException;
  public A_enumeration getByIndex(int index) throws SdaiException;
  public A_enumeration createAggregateByIndex(int index) throws SdaiException;
  public A_enumeration addAggregateByIndex(int index) throws SdaiException;
  public A_enumeration createAggregateUnordered() throws SdaiException;
  public A_enumeration getCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_enumeration createAggregateAsCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_enumeration createAggregateBefore(SdaiIterator iter) throws SdaiException;
  public A_enumeration createAggregateAfter(SdaiIterator iter) throws SdaiException;
  public void removeUnordered(A_enumeration value) throws SdaiException;
}
```

7.6.10 A_boolean

The A_boolean class shall represent aggregates of Java type boolean.

```
public class A_boolean implements Aggregate
{
  public boolean isMember(boolean value) throws SdaiException;
  public boolean getByIndex(int index) throws SdaiException;
  public void setByIndex(int index, boolean value) throws SdaiException;
  public void addByIndex(int index, boolean value) throws SdaiException;
  public void addUnordered(boolean value) throws SdaiException;
  public void removeUnordered(boolean value) throws SdaiException;
```

```
  public boolean getCurrentMember(SdaiIterator iter) throws SdaiException;
  public void setCurrentMember(SdaiIterator iter, boolean value) throws SdaiException;
  public void addBefore(SdaiIterator iter, boolean value) throws SdaiException;
  public void addAfter(SdaiIterator iter, boolean value) throws SdaiException;
}
```

7.6.11 Aa_boolean

The `Aa_boolean` class shall represent aggregates of aggregates of Java type `boolean`.

```
public class Aa_boolean implements Aggregate
{
  public boolean isMember(A_boolean value) throws SdaiException;
  public A_boolean getByIndex(int index) throws SdaiException;
  public A_boolean createAggregateByIndex(int index) throws SdaiException;
  public A_boolean addAggregateByIndex(int index) throws SdaiException;
  public A_boolean createAggregateUnordered() throws SdaiException;
  public A_boolean getCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_boolean createAggregateAsCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_boolean createAggregateBefore(SdaiIterator iter) throws SdaiException;
  public A_boolean createAggregateAfter(SdaiIterator iter) throws SdaiException;
  public void removeUnordered(A_boolean value) throws SdaiException;
}
```

7.6.12 A_binary

The `A_binary` class shall represent aggregates of Java class Binary.

```
public class A_binary implements Aggregate
{
  public boolean isMember(Binary value) throws SdaiException;
  public Binary getByIndex(int index) throws SdaiException;
  public void setByIndex(int index, Binary value) throws SdaiException;
  public void addByIndex(int index, Binary value) throws SdaiException;
  public void addUnordered(Binary value) throws SdaiException;
  public void removeUnordered(Binary value) throws SdaiException;
  public Binary getCurrentMember(SdaiIterator iter) throws SdaiException;
  public void setCurrentMember(SdaiIterator iter, Binary value) throws SdaiException;
  public void addBefore(SdaiIterator iter, Binary value) throws SdaiException;
  public void addAfter(SdaiIterator iter, Binary value) throws SdaiException;
}
```

7.6.13 Aa_binary

The `Aa_binary` class shall represent aggregates of aggregates of Java class Binary.

```
public class Aa_binary implements Aggregate
{
  public boolean isMember(A_binary value) throws SdaiException;
  public A_binary getByIndex(int index) throws SdaiException;
  public A_binary createAggregateByIndex(int index) throws SdaiException;
  public A_binary addAggregateByIndex(int index) throws SdaiException;
  public A_binary createAggregateUnordered() throws SdaiException;
```

```
  public A_binary getCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_binary createAggregateAsCurrentMember(SdaiIterator iter) throws SdaiException;
  public A_binary createAggregateBefore(SdaiIterator iter) throws SdaiException;
  public A_binary createAggregateAfter(SdaiIterator iter) throws SdaiException;
  public void removeUnordered(A_binary value) throws SdaiException;
}
```

8 Early binding

This clause defines the Java interfaces and classes for the early binding of the **SDAI_parameter_data_schema** as specified in clause 9 of ISO 10303-22 for application schemas and the SDAI_dictionary_schema schema. Early binding support of the SDAI_dictionary_schema is mandatory while early binding support for application schemas is optional. Early binding access for session entities is defined in clause 6.

8.1 Naming conventions

The names of the early binding packages, interfaces, classes, fields and methods shall be derived from the corresponding EXPRESS names of schemas, entities, attributes and defined data types in the following way: the first character of the EXPRESS name is converted to uppercase and all the other characters are converted to lowercase.

The notation **X<T>** is used to represent the Java name, constructed from a converted EXPRESS name **T** with a prefix **X**. A Java name constructed from several EXPRESS names **T1,...,Tn** is denoted by **X<T1>...<Tn>** .

EXAMPLE 1 - The following illustrates the mapping from an EXPRESS entity name to the corresponding entity interface name E<T>.
In EXPRESS:
`tHis_IS_a_sAMPLE_Name`
In Java:
`EThis_is_a_sample_name`

8.2 Schema package

Each EXPRESS schema shall be mapped to a corresponding Java package containing the schema specific Java interfaces and classes. These packages are called **schema package**s.

The schema package for the SDAI_dictionary_schema schema shall be SDAI.dictionary.

The schema package for an application schema **T** shall be **SDAI.S<T>**.

8.3 Schema class

Each EXPRESS schema shall be represented by a special public Java class within the corresponding schema package. Such a class is called **schema class**.

The schema class for the SDAI_dictionary_schema schema shall be SDictionary.

The schema class for an application schema **T** shall be **S<T>**.

A schema class may be used:

— as a parameter for the early binding version of the method `createSdaiModel`;

— as a parameter for the early binding version of the method `createSchemaInstance.`

8.4 Defined type interface

For each defined data type **T**, except those whose underlying type is SELECT or ENUMERATION, a public Java interface **E<T>** shall be available within the corresponding schema package. This interface shall extend no other interface, shall not be implemented by a Java class, and shall have no methods and no fields.

Note - Defined type interfaces are only used to distinguish between the overloaded early binding access methods for entity attributes and aggregates whose base types are select-dependent.

EXAMPLE 2 - The following example illustrates the mapping of an EXPRESS TYPE to a Java interface.

In EXPRESS:

```
SCHEMA CCC;
TYPE xxx = REAL;
END_TYPE;
END_SCHEMA;
```

In Java:

```
package SDAI.SCcc;
public interface EXxx { };
```

8.5 ENUMERATION class

Each defined data type **T**, whose underlying type is an ENUMERATION data type, shall be mapped to a public Java class **E<T>** within the corresponding schema package.

Each enumeration item shall be represented by a `public static int` data field with the name of the item in uppercase. The value of this data field shall be equal to the index of the corresponding item in `elements` list of the `enumeration_type`, specified in ISO 10303-22:6.4.28.

Each ENUMERATION class shall contain the following public static methods:

— `toInt` to convert the string representation of an enumeration item to the corresponding integer value.
public static int toInt(String str) {...};

— `toString` to convert the integer value of an enumeration item to the corresponding string representation.
public static String toString(int value) {...};

EXAMPLE 3 - The following example illustrates the Java representation of an EXPRESS enumeration.

In EXPRESS:

```
SCHEMA schema_1;

TYPE
  traffic_light = ENUMERATION OF (red, yellow, green);
END_TYPE;
```

…

In Java:

```
package SDAI.SSchema_1;

public final class ETraffic_light {
    public static final int  RED = 1;
    public static final int  YELLOW = 2;
    public static final int  GREEN = 3;

    public static int toInt(String str) {...};
    public static String toString(int value) {...};
}
```

8.6 Select data types

A value of an attribute or an aggregate member with a select dependent base type, can be represented by a value of a `primitve` type and a list of zero, one, or several defined types, see `select_value` in ISO 10303-22:9.4.8. For this part of ISO 10303, this list of defined types is called **select path**. A select path shall include no defined types, whose underlying types are select data type. A select path shall be compatible with the mapping of select values, described in ISO 10303-21 TC1, clause 11.1.8. To each select path within the set of possible select paths, for a given select dependent base type, a unique positive number shall be assigned. The number `1` shall only be used for select paths with an empty list of defined_types. A select path with an empty list of defined_types shall always have the number `1`. A possible algorithm to determine select paths is given in Annex E.

In the case that the base type of an attribute or aggregate is select-dependent, and select paths with numbers other than 1 exist, the return type of early binding methods, implementing the operations `Test attribute`, `Test current member`, and `Test by index`, shall be `int`, returning the select path number of the current value. Otherwise, the return type for these methods shall be boolean. The return values `0` and `false`, respectively, shall indicate an unset value. The select path number indicates which one of the overloaded methods for the operations `Get attribute`, `Get by index`, and `Get current member` should be used to retrieve the current value of the attribute or aggregate member.

In the case that the base type of an attribute or aggregate is select-dependent, a select path needs to be specified for the early binding methods representing the following operations:

— `Put attribute`;

— `Put by index`;

— Put current member;

— Add before current member;

— Add after current member;

— Add by index;

— Add unordered;

— Is member;

— Remove unordered;

— Get attribute;

— Create aggregate instance;

— Get by index;

— Get current member;

— Create aggregate instance by index;

— Create aggregate instance as current member;

— Create aggregate instance before current member;

— Create aggregate instance after current member;

— Add aggregate instance by index;

— Create aggregate instance unordered.

Several overloaded methods for different select paths may be needed. Additional parameters of type **E<T>**, where **T** is the name of Defined type, corresponding to the defined types of the select path, shall be added to these methods. These additional parameters are called **select parameter**s.

NOTE 1 – The select parameters enable Java to distinguish between the overloaded methods. When using such an overloaded method in an application, it is enough to pass casted null values to these select parameters. Before trying to get the value of an entity attribute or an aggregate member, an application first should determine which overloaded method has to be used, by getting the select path number with an appropriate method, implementing one of the operations `Test attribute`, `Test current member`, or `Test by index`. The select path number can be used in a Java switch statement. Depending on the actual case, the corresponding get method is chosen. To assign a value to an attribute or aggregate member, an application has to pass the value of the right Java type and, whenever select parameters are needed to specify an overloaded method, one or more appropriately casted Java null values. Examples are given in Annex D.2.

For each attribute with select-dependent base type, the Java interface for the entity, where this attribute is declared, shall contain `public static final int` data field for each select path of this select-dependent base type, except the default one with the number 1. Each such field shall be initialised with the number of the corresponding select path. The fields are named **s<A><T1>...<Tn>,** where **A** is the name of the attribute, and **T1**,.., **Tn** are the names of the defined types of the select path.

NOTE 2 - For an example of an entity interface with the data fields for select paths and with the corresponding overloaded access methods, see D.1.8. For an application example, see D.2, "switch(sbar.testbar_length(null)".

For each aggregation type whose base type is select-dependent, the corresponding aggregate class shall contain `public static final int` data field for each select path of this select-dependent base type, except the default one with the number 1. Each such field shall be initialised with the number of the corresponding select path. The fields are named **s<T1>...<Tn>,** where **T1**,.., **Tn** are the names of the defined types in the select path whose underlying type are not select types.

NOTE 3 - For an example of an aggregate class with the data fields for select paths and with the corresponding overloaded access methods, see D.4.9.

8.7 Java types for entity attributes and aggregate elements

Depending on their base type, entity attributes and aggregate elements are represented by one or several of the following Java types:

— `int` for the EXPRESS INTEGER, ENUMERATION and LOGICAL data types;

— `double` for the EXPRESS REAL and NUMBER data types;

— `boolean` for the EXPRESS BOOLEAN data type;

— `Binary` for the EXPRESS BINARY data type;

— String for the EXPRESS STRING data type;

— `EEntity` for entity data types if the base type is select-dependent and leads to one or several entity data types;

— `E<T>` if the base type is not select-dependent for a specific entity data type **T**;

— A<T> for aggregates of select-dependent data type **T**;

— A<T> for aggregates if the base type of this aggregate is not select-dependent and leads to the entity data type **T**;

— `A_binary` for aggregates if the base type of this aggregate is not select-dependent and lead to the EXPRESS BINARY data type. `Aa_binary` for an aggregate of `A_binary`;

— `A_boolean` for aggregates if the base type of this aggregate is not select-dependent and lead to the EXPRESS BOOLEAN data type. `Aa_boolean` for an aggregate of `A_boolean`;

— `A_double` for aggregates if the base type of this aggregate is not select-dependent and lead to the EXPRESS NUMBER or REAL data type. `Aa_double` for an aggregate of `A_double`;

— `A_enumeration` for aggregates if the base type of this aggregate is not select-dependent and leads to an EXPRESS ENUMERATION data types. `Aa_enumeration` for an aggregate of `A_enumeration`;

— `A_integer` for aggregates if the base type of this aggregate is not select-dependent and lead to the EXPRESS INTEGER data type. `Aa_integer` for an aggregate of `A_integer`;

— `A_string` for aggregates if the base type of this aggregate is not select-dependent and lead to the EXPRESS STRING data type. `Aa_string` for an aggregate of `A_string`.

In the case that the base type of an attribute or aggregate is a defined type whose underlying type is a defined data type or a select data type, the underlying types have to be recursively analysed.

8.8 Access methods for entity attributes

In the following subclauses on entity interface and entity class, the early binding access methods **test<T>**, **get<T>**, **unset<T>**, **set<T>**, **create<T>**, **usedin<T>**, and **attribute<T>** for entity attributes **T** are defined. These access methods have a common characteristic specified here.

Multiple inheritance of entities may cause name conflicts if several supertypes have attributes with the same name, the first parameter of every access method is used to distinguish between the entity data types where the attribute is declared. This parameter also allows distinguishing between the original attribute declaration and possible redeclarations in subtypes. The type of this parameter shall be the entity interface of the entity data type where the attribute is declared. An implementation shall not use the value of this parameter. Applications shall pass the Java null value to this parameter.

NOTE - If there is a name clash, an access method from a particular superinterface may be selected by casting the null value for the first parameter of the access method to the type of the appropriate superinterface. This mechanism allows detection of name-conflicts during compile-time.

8.9 Entity Interface

Every entity data type **T** shall be represented by a Java interface **E<T>**. In the case that an entity has one or several supertypes, its entity interface shall extend the entity interfaces of those supertypes. In the case that an entity has no supertype its entity interface shall extend EEntity.

The entity interface shall define public access methods for explicit, derived, and inverse attributes defined in this entity data type for the SDAI operations `Test Attribute`, `Get Attribute`, `Put Attribute`, `Create aggregate instance`, and `Unset Attribute value`, depending on the kind and type of the attribute. In the case that the base type of the attribute is select-dependent, `public static final int` data fields shall be added, as defined in clause 8.6.

8.9.1 Test attribute

For each explicit or derived attribute **T**, the method **test<T>** shall implement the SDAI operation `Test Attribute`, specified in ISO 10303-22:10.10.2. The return type of this method shall be `int`, if the base type of the attribute is select-dependent and select paths with numbers other than 1 exist. In all other cases the return type shall be `boolean`. The return value shall be `0` and `false`, respectively, for an unset value. The return value shall be >=1 or `true` for a set value. In the case of an int return type, the positive return value equals to the number of the select path, specifying the current attribute value.

For derived attributes, an additional parameter of type ASdaiModel, specifying the agglomerate domain, shall be added.

8.9.2 Get attribute

For each explicit, derived, or inverse attribute **T**, one or several overloaded methods **get<T>** shall implement the `Get attribute` operation, specified in ISO 10303-22:10.10.1. No method **get<T>** shall be available for redeclaring attributes except for the case that an explicit attribute is redeclaring as derived. See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the attribute.

For derived and inverse attributes, an additional parameter of type ASdaiModel, specifying the agglomerate domain, shall be added.

In addition to the error indicators, defined in ISO 10303-22:10:10.1, this method shall throw VT_NVLD, Value type invalid, in the case that the select path of the actual attribute value does not match the select path represented by the method.

8.9.3 Unset attribute value

For each explicit but not redeclaring attribute **T**, the method **unset<T>** shall implement the SDAI operation `Unset Attribute value`, specified in ISO 10303-22:10.11.4. The return type shall be void.

8.9.4 Put attribute

For each explicit but not redeclaring attribute **T**, whose base type maps to one or several Java types not being extensions of Aggregate, one or several overloaded methods **set<T>** shall implement the SDAI operation `Put Attribute`, specified in ISO 10303-22:10.11.3. The return type shall be void. The second parameter shall be the value. See clause 8.7 for the value type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the attribute.

8.9.5 Create aggregate instance

For each explicit but not redeclaring attribute **T**, whose base type maps to one or several Java types being extensions of Aggregate, one or several overloaded methods **create<T>** shall implement the SDAI operation `Create aggregate instance`, specified in ISO 10303-22:10.11.5. The method

shall return the created aggregate. See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the attribute.

8.10 Entity class

Entity classes are public Java classes **C<T>** or **C<T1>$...$<Ti>**. The entity class **C<T>** shall represent a single entity data type, a partial complex entity data type or a complex entity data type with the only one leaf entity **T**. The public class **C<T1>$...$<Ti>** shall represent a complex entity data type with the leaf entities **T1**, ..., **Ti**. The leaf entity types shall be ordered alphabetically according to their names. Differently than defined in clause 8.1, the first letter of the names of the entities **T2**, ..., **Ti** shall be written in lowercase. See ISO 10303-22:A.1.3 for further details. Entity classes shall implement the corresponding entity interfaces. Entity classes do not need to extend the corresponding Entity classes of their supertypes.

EXAMPLE 4 - The following example illustrates the mapping of an application entity to the Java programming language.

EXPRESS:
```
ENTITY XXX; ... END_ENTITY;
ENTITY YYY SUBTYPE OF (XXX) ... END_ENTITY;
```

Java:
```
public interface EXxx extends EEntity { }
public class CXxx implements EXxx extends ... { }
public class AXxx extends AEntity { ... }

public interface EYyy extends EXxx {...}

public class CYyy implements EYyy extends ... {
   ...
   public final static EEntity_definition definition = ...;
}

public class AYyy extends AEntity ... { ... }
```

8.10.1 Field definition

Every entity class shall have a convenience data field **definition** to link early binding class information to the late binding dictionary data. The Java modifiers shall be `public final static` and the type shall be `EEntitiy_definition`. The value of this field shall be the instance of EE`ntity_definition` that describes the entity corresponding to this class.

8.10.2 Get attribute definition

For each attribute **T**, the convenience method **attribute<T>** shall be available. The method shall have the Java modifiers `public static` and the return type `EAttribute`. This convenience method is used to find the corresponding instances of the entity `attribute` in the SDAI_dictionary_schema.

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

8.10.3 Find entity instance used in

For each explicit attribute **T**, which base type is an entity, or contains at least one entity type in an aggregation type, or leads to at least one entity type through a SELECT type, or any combination of the previous two, the `public static` method **usedin<T>** shall implement the SDAI operation `Find entity instance usedin`, specified in ISO 10303-22:10.10.9.

The second parameter is the entity instance whose users are requested. If the type of the attribute is an entity data type or a, possibly nested, aggregate of a specific entity data type, then the parameter type shall be the corresponding entity interface. Otherwise, the parameter type shall be `EEntity`.

The third parameter shall be of type `ASdaiModel`, specifying the agglomerate domain of entity instances to check as users of the specified instance, given by the second parameter. An application may pass a null value to specify that the domain is the SdaiModel, owning the entity instance, given by the second parameter.

The fourth parameter shall have the type `AEntity` for the `Result`.

The method shall return an `int` value, specifying the number of instances added to `Result`.

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

8.11 Aggregate class

Public classes for early binding aggregates shall be available for the following three cases.

— For each entity interface **E<T>** there shall be a corresponding public aggregate class **A<T>** implementing `Aggregate`.

— For each EXPRESS select type **T**, used as base type of an aggregate, there shall be a corresponding public aggregate class **A<T>** implementing `Aggregate`.

— Further aggregate classes shall be available for nested aggregates of the above two cases, as needed. The name of the public aggregate class is **Aa<T>** for the second nesting level, **Aaa<T>** for the third nesting level and so on.

In the following subclauses, ETYPE and ATYPE represent aggregate members. **ETYPE** is used to represent E<T> or the types `String`, `Binary`, `boolean`, `int` or `double`. **ATYPE** is used to represent either ETYPE or A<T> or `A_string`, `A_double`, `A_integer`, `A_binary`, `A_enumeration`, `A_boolean`, or any nested aggregate. The notation "... /*select parameters */" is used to indicate 0, 1, or several select parameters as defined in clause 8.6.

8.11.1 Is member

The `isMember` method shall implement the SDAI operation `Is member`, specified in ISO 10303-22:10.12.2

```
boolean isMember(ATYPE value, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the value type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.2 Test by index

Every aggregate shall have a method `testByIndex`, implementing the operation `Test By Index`, specified in ISO 10303-22:10.17.1. The method shall take one parameter of type `int` which shall specify the index of the member of the aggregate to be tested. In addition to ARRAYs, this method shall apply to LISTs.

The return type of this method shall be `int`, if the base type of the aggregate is select-dependent and select paths with numbers other than 1 exist. In all other cases the return type shall be `boolean`. The return value shall be `0` and `false`, respectively, for an unset value. The return value shall be >=1 or `true` for a set value. In case of an int return type, the positive return value equals to the number of the select path specifying the current value of the aggregate member. See clause 8.6 for more details.

8.11.3 Get by index

The `getByIndex` method shall implement the SDAI operation `Get by index`, specified in ISO 10303-22:10.15.1.

```
ATYPE getByIndex(int index, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.4 Put by index

The `setByIndex` method shall implement the SDAI operation `Put by index`, specified in ISO 10303-22:10.16.1.

```
void setByIndex(int index, ETYPE value, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the value type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.5 Create aggregate instance by index

The `creatAggregateByIndex` method shall implement the SDAI operation `Create aggregate instance by index`, specified in ISO 10303-22:10.16.2.

```
<AT> createAggregateByIndex(int index, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.6 Add by index

The `addByIndex` method shall implement the SDAI operation `Add by index`, specified in ISO 10303-22:10.19.3.

```
void addByIndex(int index, ETYPE value, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the value type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.7 Add aggregate instance by index

The `addAggregateByIndex` method shall implement the SDAI operation `Add aggregate instance by index`, specified in ISO 10303-22:10.19.6.

```
<AT> addAggregateByIndex(int index, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.8 Test current member

Each aggregate shall have a method testCurrentMember, implementing the operation `Test Current Member`, specified in ISO 10303-22:10.17.2. The method shall take one parameter of type SdaiIterator which shall specify the member of the aggregate to be tested. This method shall also apply to LISTs, BAGs, and SETs.

The return type of this method shall be `int`, if the base type of the aggregate is select-dependent and select paths with numbers other than 1 exist. In all other cases the return type shall be `boolean`. The return value shall be `0` and `false`, respectively for an unset value. The return value shall be >=1 or `true` for a set value. In case of an int return type, the positive return value equals to the number of the select path specifying the current value of the aggregate member. See clause 8.6 for more details.

8.11.9 Get current member

The `getCurrentMember` method shall implement the SDAI operation `Get current member`, specified in ISO 10303-22:10.12.7.

```
ATYPE getCurrentMember(SdaiIterator iter, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

This method shall implement the following error indicator, which is in addition to those defined in ISO 10303-22: VT_NVLD, Value type invalid.

8.11.10 Put current member

The `setCurrentMember` method shall implement the SDAI operation `Put current member`, specified in ISO 10303-22:10.13.2.

```
void setCurrentMember(SdaiIterator iter, ETYPE value, ... /*select parameters */)
    throws SdaiException;
```

See clause 8.7 for the value type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.11 Create aggregate instance as current member

The `createAggregateMember` method shall implement the SDAI operation `Create aggregate instance as current member`, specified in ISO 10303-22:10.3.1.

```
<AT> createAggregateAsCurrentMember(SdaiIterator iter, ... /*select parameters */)
    throws SdaiException;
```

See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.12 Add before current member

The `addBefore` method shall implement the SDAI operation `Add before current member`, specified in ISO 10303-22:10.19.1.

```
void addBefore(SdaiIterator iter, ETYPE value, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the value type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.13 Create aggregate instance before current member

The `createAggregateBefore` method shall implement the SDAI operation `Create aggregate instance before current member`, specified in ISO 10303-22:10.19.4.

```
<AT> createAggregateBefore(SdaiIterator iter, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.14 Add after current member

The `addAfter` method shall implement the SDAI operation `Add after current member`, specified in ISO 10303-22:10.19.2.

```
void addAfter(SdaiIterator iter, ETYPE value, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the value type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.15 Create aggregate instance after current member

The `createAggregateAfter` method shall implement the SDAI operation `Create aggregate instance after current member`, specified in ISO 10303-22:10.19.5.

```
<AT> createAggregateAfter(SdaiIterator iter, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.16 Add unordered

The `addUnordered` method shall implement the SDAI operation `Add unordered`, specified in ISO 10303-22:10.14.1.

```
void addUnordered(ETYPE value, ... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the value type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.17 Create aggregate instance unordered

The `createAggregateUnordered` method shall implement the SDAI operation `Create aggregate instance unordered`, specified in ISO 10303-22:10.14.2.

```
<AT> createAggregateUnordered(... /*select parameters */) throws SdaiException;
```

See clause 8.7 for the return type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

8.11.18 Remove unordered

The `removeUnordered` method shall implement the SDAI operation `Remove unordered`, specified in ISO 10303-22:10.14.3.

```
void removeUnordered(ATYPE value, ... /*select parameters */) throws SdaiException;
}
```

See clause 8.7 for the value type and clause 8.6 about overloaded methods in the case of a select-dependent base type of the aggregate.

9 Error handling

SDAI errors, specified in ISO 10303-22 clause 11, which may occur while executing methods implementing SDAI operations, specified in ISO 10303-22 clause 10, are handled by the Java

exception mechanism. An instance of the class **SdaiException** defined below shall be thrown in the case when an error occurs. This instance may be caught by an application program.

Because a **SdaiException** is thrown under all defined circumstances, there is no need to map the following concepts from ISO 10303-22 to this part of ISO 10303:

— entity **error_event**, specified in ISO 10303-22:7.4.7;

— entity **event**, specified in ISO 10303-22:7.4.6;

— the select type **error_base**, specified in ISO 10303-22:7.3.2;

— operation **Record error**, specified in ISO 10303-22:10.4.1;

— operation **Start event recording**, specified in ISO 10303-22:10.4.2;

— operation **Stop event recording**, specified in ISO 10303-22:10.4.3.

9.1 SdaiException

The SDAI standard error indicator constants shall be declared in the Java class SDAI.lang.SdaiException.

The following error indicators shall be added to those specified in ISO 10303-22 clause 11:

— RP_DUP, repository duplicate,

— RP_NCTE, repository not compatible with clear text encoding.

The error base for RP_DUP shall be SdaiRepository, and there shall be no error base for the error indicator RP_NCTE.

The `SdaiException` class shall be defined as follows:

```
package SDAI.lang;
import Hashtable;

public final class SdaiException extends Exception {

  public static final int SS_OPN = 10;
  public static final int SS_NAVL = 20;
  public static final int SS_NOPN = 30;
  public static final int RP_NEXS = 40;
  public static final int RP_NAVL = 50;
  public static final int RP_OPN = 60;
  public static final int RP_NOPN = 70;
  public static final int RP_DUP = 75;
  public static final int TR_EAB = 80;
  public static final int TR_EXS = 90;
  public static final int TR_NAVL = 100;
```

```
  public static final int TR_RW = 110;
  public static final int TR_NRW = 120;
  public static final int TR_NEXS = 130;
  public static final int MO_NDEQ = 140;
  public static final int MO_NEXS = 150;
  public static final int MO_NVLD = 160;
  public static final int MO_DUP = 170;
  public static final int MX_NRW = 180;
  public static final int MX_NDEF = 190;
  public static final int MX_RW = 200;
  public static final int MX_RO = 210;
  public static final int SD_NDEF = 220;
  public static final int ED_NDEF = 230;
  public static final int ED_NDEQ = 240;
  public static final int ED_NVLD = 250;
  public static final int RU_NDEF = 260;
  public static final int EX_NSUP = 270;
  public static final int AT_NVLD = 280;
  public static final int AT_NDEF = 290;
  public static final int SI_DUP = 300;
  public static final int SI_NEXS = 310;
  public static final int EI_NEXS = 320;
  public static final int EI_NAVL = 330;
  public static final int EI_NVLD = 340;
  public static final int EI_NEXP = 350;
  public static final int AI_NEXS = 380;
  public static final int AI_NVLD = 390;
  public static final int AI_NSET = 400;
  public static final int VA_NVLD = 410;
  public static final int VA_NEXS = 420;
  public static final int VA_NSET = 430;
  public static final int VT_NVLD = 440;
  public static final int IR_NEXS = 450;
  public static final int IR_NSET = 460;
  public static final int IX_NVLD = 470;
  public static final int ER_NSET = 480;
  public static final int OP_NVLD = 490;
  public static final int FN_NAVL = 500;
  public static final int RP_NCTE = 530;
  public static final int SY_ERR = 1000;

  public SdaiException() {
    super((String) error_descriptions.get(new Integer(SY_ERR)));
  };

  public SdaiException(int id) {
    super((String) error_descriptions.get(new Integer(id)));
    error_id = id;
  };

  public SdaiException(int id, Object base) {
    super((String) error_descriptions.get(new Integer(id)));
    error_id = id;
```

```
    error_base = base;
  };

  public SdaiException(int id, Exception exception) {
    super((String) error_descriptions.get(new Integer(id)));
    error_id = id;
    underlyingException = exception;
  };

  public SdaiException(int id, Object base, Exception exception) {
    super((String) error_descriptions.get(new Integer(id)));
    error_id = id;
    error_base = base;
    underlyingException = exception;
  };

  public Exception getUnderlyingException() {
    return underlyingException;
  };

  public int getErrorId() {
    return error_id;
  };

  public Object getErrorBase() {
    return error_base;
  };

  public static String getErrorDescription(int id) {
    return (String) error_descriptions.get(new Integer(id));
  };

  public String toString() {
    if (underlyingException == null) {
      return super.toString();
    } else {
       return super.toString() + "\nUnderlying Exception:  " + underlyingException.toString();
    }
  };

  // private fields and methods
  private int error_id = 1000;
  private Object error_base = null;
  private Exception underlyingException = null;
  private static Hashtable error_descriptions = new Hashtable();

 private static void initError_descriptions() {
   error_descriptions.put(new Integer(SS_OPN),
    "SS_OPN - Session Open");
   error_descriptions.put(new Integer(SS_NAVL),
    "SS_NAVL - Session not Available");
   error_descriptions.put(new Integer(SS_NOPN),
    "SS_NOPN - Session is not open");
   error_descriptions.put(new Integer(RP_NEXS),
```

```
    "RP_NEXS - Repository does not exist");
  error_descriptions.put(new Integer(RP_NAVL),
    "RP_NAVL - Repository not available");
  error_descriptions.put(new Integer(RP_OPN),
    "RP_OPN - Repository open");
  error_descriptions.put(new Integer(RP_NOPN),
    "RP_NOPN - Repository is not open");
  error_descriptions.put(new Integer(RP_DUP),
    "RP_DUP - Repository duplicate");
  error_descriptions.put(new Integer(TR_EAB),
    "TR_EAB - Transaction ended abnormally");
  error_descriptions.put(new Integer(TR_EXS),
    "TR_EXS - Transaction exists");
  error_descriptions.put(new Integer(TR_NAVL),
    "TR_NAVL - Transaction currently not available");
  error_descriptions.put(new Integer(TR_RW),
    "TR_RW - Transaction read-write");
  error_descriptions.put(new Integer(TR_NRW),
    "TR_NRW - Transaction not read-write");
  error_descriptions.put(new Integer(TR_NEXS),
    "TR_NEXS - Transaction does not exist");
  error_descriptions.put(new Integer(MO_NDEQ),
    "MO_NDEQ - SDAI-model not domain equivalent");
  error_descriptions.put(new Integer(MO_NEXS),
    "MO_NEXS - SDAI-model does not exist");
  error_descriptions.put(new Integer(MO_NVLD),
    "MO_NVLD - SDAI-model invalid");
  error_descriptions.put(new Integer(MO_DUP),
    "MO_DUP - SDAI-model duplicate");
  error_descriptions.put(new Integer(MX_NRW),
    "MX_NRW - SDAI-model access not read-write");
  error_descriptions.put(new Integer(MX_NDEF),
    "MX_NDEF - SDAI-model access not defined");
  error_descriptions.put(new Integer(MX_RW),
    "MX_RW - SDAI-model access read-write");
  error_descriptions.put(new Integer(MX_RO),
    "MX_RO - SDAI-model access read-only");
  error_descriptions.put(new Integer(SD_NDEF),
    "SD_NDEF - Schema definition not defined");
  error_descriptions.put(new Integer(ED_NDEF),
    "ED_NDEF - Entity definition not defined");
  error_descriptions.put(new Integer(ED_NDEQ),
    "ED_NDEQ - Entity definition not domain equivalent");
  error_descriptions.put(new Integer(ED_NVLD),
    "ED_NVLD - Entity definition invalid");
  error_descriptions.put(new Integer(RU_NDEF),
    "RU_NDEF - Rule not defined");
  error_descriptions.put(new Integer(EX_NSUP),
    "EX_NSUP - Expression evaluation not supported");
  error_descriptions.put(new Integer(AT_NVLD),
    "AT_NVLD - Attribute invalid");
  error_descriptions.put(new Integer(AT_NDEF),
    "AT_NDEF - Attribute not defined");
```

```
    error_descriptions.put(new Integer(SI_DUP),
     "SI_DUP - Schema instance duplicate");
    error_descriptions.put(new Integer(SI_NEXS),
     "SI_NEXS - Schema instance does not exist");
    error_descriptions.put(new Integer(EI_NEXS),
     "EI_NEXS - Entity instance does not exist");
    error_descriptions.put(new Integer(EI_NAVL),
     "EI_NAVL - Entity instance not available");
    error_descriptions.put(new Integer(EI_NVLD),
     "EI_NVLD - Entity instance invalid");
    error_descriptions.put(new Integer(EI_NEXP),
     "EI_NEXP - Entity instance not exported");
    error_descriptions.put(new Integer(AI_NEXS),
     "AI_NEXS - Aggregate isntance does not exist");
    error_descriptions.put(new Integer(AI_NVLD),
     "AI_NVLD - Aggregate instance invalid");
    error_descriptions.put(new Integer(AI_NSET),
     "AI_NSET - Aggregate instance is empty");
    error_descriptions.put(new Integer(VA_NVLD),
     "VA_NVLD - Value invalid");
    error_descriptions.put(new Integer(VA_NEXS),
     "VA_NEXS - Value does not exist");
    error_descriptions.put(new Integer(VA_NSET),
     "VA_NSET - Value not set");
    error_descriptions.put(new Integer(VT_NVLD),
     "VT_NVLD - Value type invalid");
    error_descriptions.put(new Integer(IR_NEXS),
     "IR_NEXS - Iterator does not exist");
    error_descriptions.put(new Integer(IR_NSET),
     "IR_NSET - Current member is not defined");
    error_descriptions.put(new Integer(IX_NVLD),
     "IX_NVLD - Index invalid");
    error_descriptions.put(new Integer(ER_NSET),
     "ER_NSET - Event recording not set");
    error_descriptions.put(new Integer(OP_NVLD),
     "OP_NVLD - Operator invalid");
    error_descriptions.put(new Integer(FN_NAVL),
     "FN_NAVL - Function not available");
    error_descriptions.put(new Integer(RP_NCTE),
     "RP_NCTE - Repository not compatible with clear text encoding");
    error_descriptions.put(new Integer(SY_ERR),
     "SY_ERR - Underlying system error");
};

  static {
    initError_descriptions();
  };
};
```

10 SDAI dictionary

The elements of the SDAI_dictionary_schema, specified in ISO 10303-22:6 shall be mapped to early binding Java interfaces and classes as defined in clause 8 with the following changes:

— the Java package for the interfaces and classes is `SDAI.dictionary`;

— there are additional SDAI methods for the entity interface `EEntity_definition` as defined below;

10.1 Entity_definition

The `EEntity_definition` interface shall represent the SDAI entity **entity_definition**, specified in ISO 10303-22:6.4.12. In addition to the definitions in subclause 8.6, `EEntity_definition` shall contain two further SDAI methods.

```
package SDAI.dictionary;

public interface EEntity_definition ...{
  ...
  boolean isSubtypeOf(EEntity_definition compType) throws SdaiException;
  boolean isDomainEquivalentWith(EEntity_definition compType) throws SdaiException;
}
```

10.1.1 Is subtype of

The `isSubtypeOf` method shall implement the SDAI operation `Is subtype of` and SDAI operation `Is SDAI subtype of`, specified in ISO 10303-22:10.9.2 and ISO 10303-22:10.9.3, respectively.

10.1.2 Is domain equivalent with

The `isDomainEquivalentWith` method shall implement the SDAI operation `Is domain equivalent with`, specified in ISO 10303-22:10.9.4.

Annex A
(normative)

Information object registration

To provide for unambiguous identification of an information object in an open system, the object identifier

{ iso standard 10303 part(27) version (1) }

is assigned to this part of ISO 10303. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

Annex B
(normative)

Mapping between Part-21 and SDAI

Both, part-21 and the SDAI, define implementation methods of STEP. This annex defines a mapping between the elements in a part-21 exchange structure and the elements in SDAI for the purpose of the methods imporClearTextEncoding, clause 6.1.8, and exportClearTextEncoding, clause 6.3.7.

This language binding to the SDAI has been defined such, that every exchange structure can be imported into SDAI. For this additional attributes have been added to the `sdai_repository` for time_stamp, author, organization, preprocessor_version, originating_system and authorization in order to make the `sdai_repository` compatible with part-21. However due to limitations in part-21 not every SDAI population can be exported to a part-21 exchange structure.

The mapping between part-21 and SDAI is defined as follows:

— Both part-21 and SDAI define explicitly how entity instances and their attribute values are represented. A part-21 `entity instance name` and an `SDAI persistent identifier` are mapped to each other;

— A part-21 `data section` and an `SDAI-model` are mapped to each other. During the import of a clear text encoding, the name of the `sdai_model` instance shall be set to “default” and no schema_instances shall be available;

— A whole part-21 exchange structure and an SDAI-repository are mapped to each other. The part-21 file_name is mapped to the name of the SdaiRepository;

— The SDAI schema-instance is not mapped to the part-21 exchange structure;

— SdaiRepositories, containing entity instances which references entity instances in other SdaiRepositories cannot be mapped to a part-21 exchange structure.

The `SdaiSession` method `importClearTextEncoding` creates a new `SdaiRepository`, containing one `SdaiModel` with the name `model1` and one `SchemaInstance` with the name `schema1`. The `SdaiModel` is associated with the `SchemaInstance`. The native schema of the `SchemaInstance` and the underlying schema of the `SdaiModel` are set to the FILE_SCHEMA of the part-21 exchange structure.

The SdaiRepository method `exportClearTextEncoding` requires exactly one instance of a `SdaiModel` and no instance of a `SchemaInstance` in this `SdaiRepository`. All references between application instances of this `SdaiModel` shall be local to this `SdaiModel`, otherwise the SdaiException RP_NCTE, repository not compatible with clear text encoding, is thrown.

The instance names used inside the data section of the clear text encoding shall be used as the persistent labels of SDAI application instances. For the SdaiRepository method `exportClearTextEncoding`, the format of the persistent labels shall be that of the instance names as specified in ISO 10303-21:4.4.2. Otherwise, the SdaiException RP_NCTE shall be thrown.

Annex C
(informative)

Events

The event mechanism is a widely used mechanism in various Java APIs. It allows listener object to register and de-register at source objects. Registered objects can receive event notifications. This event mechanism is adopted for the purpose of this part of ISO 10303 through the `SdaiEvent` class and the `SdaiListener` and `SdaiEventSource` interfaces. Implementations may support the event mechanism by extending or implementing `SdaiEventSource` for either one or several of `SdaiSession`, `SdaiRepository`, `SdaiModel`, `SchemaInstance`, `EEntity` and `Aggregate`.

C.1 SdaiEvent

Objects of type `SdaiEvent` are send to the notification methods of registered SdaiEventListeners in case of any change of the object. An SdaiEvent with the id `INVALID` signals that the object is going to become invalid. An SdaiEvent with the id `MODIFIED` signals that one or several of the attributes of the object changed.

```
package SDAI.lang;
import SDAI.dictionary.*;
import java.util.EventObject;

public class SdaiEvent extends EventObject {
  public static final int INVALID;
  public static final int MODIFIED;

  public int getId();
  public Object getSource();
}
```

The method `getId` returns the id of the event, which is either `INVALID` or `MODIFIED`.
The method `getSource` returns the object from which this event comes from. The type of the object is either `SdaiSession`, `SdaiRepository`, `SdaiModel`, `SchemaInstance`, `EEntity` or `Aggregate`.

C.2 SdaiListener

Application classes implementing `SdaiListener` are able to receive `SdaiEvents` through the method `actionPerformed`.

```
package SDAI.lang;
import java.util.EventListener;

public interface SdaiListener extends EventListener {
  public void actionPerformed(SdaiEvent e);
}
```

C.3 SdaiEventSource

Applications objects, implementing `SdaiListener` are able to register and de-register on SDAI objects supporting `SdaiEventSource`. With the method `addSdaiListener` an application objects registers on an SDAI object to receive `SdaiEvents` from this SDAI object. The method `removeSdaiListener` de-registers previously registered application objects to no longer receive `SdaiEvents` from this SDAI object.

```
package SDAI.lang;

public interface SdaiEventSource {
  void addSdaiListener(SdaiListener listener);
  void removeSdaiListener(SdaiListener listener);
}
```

Annex D
(informative)

Examples

D.1 Mass measure schema

```
EXAMPLE 1, taken from ISO10303-21 technical corrigendum EXAMPLE 26

SCHEMA mass_measure_schema;

TYPE Mass = SELECT (Mass_Spec, Mass_Substitute);
END_TYPE;

TYPE Mass_Spec = SELECT (Measured_Mass, Computed_Mass, Estimated_Mass);
END_TYPE;

TYPE Measured_Mass = REAL;
END_TYPE;

TYPE Computed_Mass = Extended_Real;
END_TYPE;

TYPE Estimated_Mass = REAL;
END_TYPE;

TYPE Mass_Substitute = SELECT(Weight, Estimated_Mass);
END_TYPE;

TYPE Weight = REAL;
END_TYPE;

TYPE Extended_Real = SELECT (FloatingNumber, NotaNumber);
END_TYPE;

TYPE FloatingNumber = REAL(7);
END_TYPE;

TYPE NotaNumber = ENUMERATION OF (plus_infinity, minus_infinity, indeterminate, invalid);
END_TYPE;

ENTITY Steel_Bar;
  bar_length: Extended_Real;
  bar_mass:   Mass;
END_ENTITY;

END_SCHEMA;
```

The resulting interfaces and classes are as follows.

No interfaces are given for Extended_Real, Mass, Mass_Spec, Mass_Substitute, as they are not needed to define the early binding access methods.

D.1.1 SMass_measure_schema.java

Java class representing the EXPRESS schema mass_measure_schema.

```
package SDAI.SMass_measure_schema;

public class SMass_measure_schema {
...
}
```

D.1.2 EMeasured_mass.java

Java interface representing the defined data type measured_mass.

```
package SDAI.SMass_measure_schema;

public interface EMeasured_mass {
}
```

D.1.3 EComputed_mass.java

Java interface representing the defined data type computed_mass.

```
package SDAI.SMass_measure_schema;

public interface EComputed_mass {
}
```

D.1.4 EEstimated_mass.java

Java interface representing the defined data type estimated_mass.

```
package SDAI.SMass_measure_schema;

public interface EEstimated_mass {
}
```

D.1.5 EWeight.java

Java interface representing the defined data type weight.

```
package SDAI.SMass_measure_schema;

public interface EWeight {
}
```

D.1.6 EFloatingnumber.java

Java interface representing the defined data type floatingnumber.

```
package SDAI.SMass_measure_schema;

public interface EFloatingnumber {
}
```

D.1.7 ENotanumber.java

Java class representing the enumeration data type notanumber.

```
package SDAI.SMass_measure_schema;

public class ENotanumber {

  public static final int PLUS_INFINITY; // plus_infinity
  public static final int MINUS_INFINITY; // minus_infinity
  public static final int INDETERMINATE; // indeterminate
  public static final int INVALID; // invalid

  public static int toInt(String str);

  public static String toString(int value);
};
```

D.1.8 ESteel_bar.java

Java interface implementing the entity data type steel_bar.

```
package SDAI.SMass_measure_schema;

public interface ESteel_bar extends SDAI.lang.EEntity {

    // constants and methods for SELECT attribute: bar_length
    int sBar_lengthFloatingnumber = ...;
    int sBar_lengthNotanumber = ...;
    int testBar_length(ESteel_bar type)
        throws SDAI.lang.SdaiException;
    double getBar_length(ESteel_bar type, EFloatingnumber node1)
        throws SDAI.lang.SdaiException; // case sBar_lengthFloatingnumber
    int getBar_length(ESteel_bar type, ENotanumber node1)
        throws SDAI.lang.SdaiException; // case sBar_lengthNotanumber
    void setBar_length(ESteel_bar type, double value, EFloatingnumber node1)
        throws SDAI.lang.SdaiException; // case sBar_lengthFloatingnumber
    void setBar_length(ESteel_bar type, int value, ENotanumber node1)
        throws SDAI.lang.SdaiException; // case sBar_lengthNotanumber
    void unsetBar_length(ESteel_bar type)
        throws SDAI.lang.SdaiException;

    // constants and methods for SELECT attribute: bar_mass
    int sBar_massMeasured_mass = ...;
    int sBar_massComputed_massFloatingnumber = ...;
    int sBar_massComputed_massNotanumber = ...;
    int sBar_massEstimated_mass = ...;
    int sBar_massWeight = ...;
```

```
    int testBar_mass(ESteel_bar type)
        throws SDAI.lang.SdaiException;
    double getBar_mass(ESteel_bar type, EMeasured_mass node1)
        throws SDAI.lang.SdaiException; // case sBar_massMeasured_mass
    double getBar_mass(ESteel_bar type, EComputed_mass node1, EFloatingnumber node2)
        throws SDAI.lang.SdaiException; // case sBar_massComputed_massFloatingnumber
    int getBar_mass(ESteel_bar type, EComputed_mass node1, ENotanumber node2)
        throws SDAI.lang.SdaiException; // case sBar_massComputed_massNotanumber
    double getBar_mass(ESteel_bar type, EEstimated_mass node1)
        throws SDAI.lang.SdaiException; // case sBar_massEstimated_mass
    double getBar_mass(ESteel_bar type, EWeight node1)
        throws SDAI.lang.SdaiException; // case sBar_massWeight
    void setBar_mass(ESteel_bar type, double value, EMeasured_mass node1)
        throws SDAI.lang.SdaiException; // case sBar_massMeasured_mass
    void setBar_mass(ESteel_bar type, double value, EComputed_mass node1, EFloatingnumber node2)
        throws SDAI.lang.SdaiException; // case sBar_massComputed_massFloatingnumber
    void setBar_mass(ESteel_bar type, int value, EComputed_mass node1, ENotanumber node2)
        throws SDAI.lang.SdaiException; // case sBar_massComputed_massNotanumber
    void setBar_mass(ESteel_bar type, double value, EEstimated_mass node1)
        throws SDAI.lang.SdaiException; // case sBar_massEstimated_mass
    void setBar_mass(ESteel_bar type, double value, EWeight node1)
        throws SDAI.lang.SdaiException; // case sBar_massWeight
    void unsetBar_mass(ESteel_bar type) throws SDAI.lang.SdaiException;
}
```

D.1.9 CSteel_bar.java

Java class implementing the complex entity data type steel_bar.

```
package SDAI.SMass_measure_schema;

public class CSteel_bar implements ESteel_bar extends ... {
    public static final SDAI.dictionary.EEntity_definition definition = ...;
    ...
    public static SDAI.dictionary.EAttribute attributeBar_length(ESteel_bar type)
        throws SdaiException { ... }
    public static SDAI.dictionary.EAttribute attributeBar_mass(ESteel_bar type)
        throws SdaiException { ... }
}
```

D.2 Mass measure schema - early binding access

Application with early binding access for the Mass-Measure schema as given in D.1.
The following instances are created and listed on the screen:

```
#1 = STEEL_BAR(FLOATINGNUMBER(77.0), MEASURED_MASS(13.25));
#2 = STEEL_BAR(NOTANUMBER(.INDETERMINATE.), ESTIMATED_MASS(10.0));
#3 = STEEL_BAR(FLOATINGNUMBER(77.0),
        COMPUTED_MASS(FLOATINGNUMBER(14.77719)));
```

File MassMeasureEarly.java:

```
import SDAI.lang.*;
import SDAI.SMass_measure_schema.*;

public class MassMeasureEarly {

    public static final void main(String argv[]) throws SdaiException {

        SdaiSession session = SdaiSession.openSession();

        SdaiTransaction transaction = session.startTransactionReadWriteAccess();

        SdaiRepository rep = session.createRepository("Steel_warehouse", null);
        rep.openRepository();


        // *********** creation of entity-instances **************
        SdaiModel mod = rep.createSdaiModel("example_2", SMass_measure_schema.class);

        mod.startReadWriteAccess();
        ESteel_bar sb1 = (ESteel_bar) mod.createEntityInstance(CSteel_bar.class);
        sb1.setBar_length(null, 77.00, (EFloatingnumber)null);
        sb1.setBar_mass(null, 13.25, (EMeasured_mass)null);

        ESteel_bar sb2 = (ESteel_bar) mod.createEntityInstance(CSteel_bar.class);
        sb2.setBar_length(null, ENotanumber.INDETERMINATE,
           (ENotanumber)null);
        sb2.setBar_mass(null, 10., (EEstimated_mass)null);

        ESteel_bar sb3 = (ESteel_bar) mod.createEntityInstance(CSteel_bar.class);
        sb3.setBar_length(null, 77.00, (EFloatingnumber)null);
        sb3.setBar_mass(null, 14.7719, (EComputed_mass)null,
           (EFloatingnumber)null);


        // *********** searching for entity-instances **************
        Aggregate insts = mod.getEntityExtentInstances(ESteel_bar.class);
        SdaiIterator it = insts.createIterator();
        while (it.next()) {
           ESteel_bar sbar = (ESteel_bar)insts.getCurrentMemberObject(it);
           double length, mass;
           int not_length, not_mass;


           String attributes = "Attributes of this steel bar: length = ";

           switch(sbar.testBar_length(null)) {
           case 0:
                  attributes += "unset";
                  break;
           case ESteel_bar.sBar_lengthFloatingnumber:
                  length = sbar.getBar_length(null, (EFloatingnumber)null);
                  attributes += length;
                  break;
           case ESteel_bar.sBar_lengthNotanumber:
```

```
                not_length = sbar.getBar_length(null, (ENotanumber)null);
                attributes += ENotanumber.toString(not_length);
                break;
            default:
                attributes += "unknown type";
                break;
            }

            attributes += ", mass = ";
            switch(sbar.testBar_mass(null)) {
            case 0:
                attributes += "unset";
                break;
            case ESteel_bar.sBar_massMeasured_mass:
                mass = sbar.getBar_mass(null, (EMeasured_mass)null);
                attributes += mass;
                break;
            case ESteel_bar.sBar_massComputed_massFloatingnumber:
                mass = sbar.getBar_mass(null, (EComputed_mass)null,
                    (EFloatingnumber)null);
                attributes += mass;
                break;
            case ESteel_bar.sBar_massComputed_massNotanumber:
                not_mass = sbar.getBar_mass(null, (EComputed_mass)null,
                    (ENotanumber)null);
                attributes += ENotanumber.toString(not_mass);
                break;
            case ESteel_bar.sBar_massEstimated_mass:
                mass = sbar.getBar_mass(null, (EEstimated_mass)null);
                attributes += mass;
                break;
            case ESteel_bar.sBar_massWeight:
                mass = sbar.getBar_mass(null, (EWeight)null);
                attributes += mass;
                break;
            default:
                attributes += "unknown type";
                break;
            }
            System.out.println(attributes);
        }
        it.beginning();
        System.out.println("\nThe same instances printed in ISO 10303-21 style:");
        while (it.next()) {
            ESteel_bar sbar = (ESteel_bar)insts.getCurrentMemberObject(it);
            System.out.println(sbar.toString());
        }
        session.closeSession();
    } // main
}
```

D.3 Mass measure schema - late binding access

This example is equivalent to example D.2 above but with late binding access to application instances.

File MassMeasureLate.java:

```
import SDAI.lang.*;
import SDAI.dictionary.*;
import SDAI.SMass_measure_schema.*;

public class MassMeasureLate {

    public static final void main(String argv[]) throws SdaiException {
        SdaiSession session = SdaiSession.openSession();

        SdaiTransaction transaction = session.startTransactionReadWriteAccess();


        // find the schema_definition for Mass_measure_schema
        ESchema_definition exp1_schema = null;
        SdaiModel exp1_dic_model = null;
        SchemaInstance si = session.getDataDictionary();
        ASdaiModel set_of_associated_models = si.getAssociatedModels();
        SdaiIterator it1 = set_of_associated_models.createIterator();
        while (it1.next()) {
            SdaiModel model = set_of_associated_models.getCurrentMember(it1);
            String m_name = model.getName();
            if (m_name.equals("MASS_MEASURE_SCHEMA_DICTIONARY_DATA")) {
                   exp1_dic_model = model;
                   if (exp1_dic_model.getMode() == SdaiModel.NO_ACCESS) {
                          exp1_dic_model.startReadOnlyAccess();
                   }
                   exp1_schema = exp1_dic_model.getDefinedSchema();
                   break;
            }
        }
        if (exp1_dic_model == null) {
            System.out.println("Dictionary data model for Mass_measure_schema schema not found");
            return;
        }

        // find entity_definition for steel_bar
        EEntity_definition steel_bar_def = null;
        AEntity_definiton entities = exp1_schema.getEntities(null, null);
        entities.attachIterator(it1);
        while (it1.next()) {
            EEntity_definition ed = entities.getCurrentMember(it1);
            String name = ed.getName(null);
            if (name.equalsIgnoreCase("STEEL_BAR")) {
                   steel_bar_def = ed;
                   break;
            }
        }
        if (steel_bar_def == null) {
```

```
        System.out.println("Steel bar not found");
        return;
    }

    // find attributes bar_length and bar_mass
    EAttribute bar_length = null;
    EAttribute bar_mass = null;
    AAttribute attributes = steel_bar_def.getAttributes(null, null);
    attributes.attachIterator(it1);
    while (it1.next()) {
        EAttribute att = attributes.getCurrentMember(it1);
        String name = att.getName(null).toUpperCase();
        if (name.equals("BAR_LENGTH")) {
               bar_length = att;
        } else
        if (name.equals("BAR_MASS")) {
               bar_mass = att;
        }
    }
    if (bar_length == null || bar_mass == null) {
        System.out.println("No bar length or mass");
        return;
    }

    // find some defined_types
    EDefined_type dtFloatingNumber = null;
    EDefined_type dtNotANumber = null;
    EDefined_type dtMeasuredMass = null;
    EDefined_type dtEstimatedMass = null;
    EDefined_type dtComputedMass = null;
    EDefined_type dtWeight = null;
    ADefined_types types = exp1_schema.getTypes(null, null);
    types.attachIterator(it1);
    while (it1.next()) {
        EDefined_type dt = types.getCurrentMember(it1);
        String name = dt.getName(null).toUpperCase();
        if (name.equals("MEASURED_MASS")) {
               dtMeasuredMass = dt;
        } else
        if (name.equals("COMPUTED_MASS")) {
               dtComputedMass = dt;
        } else
        if (name.equals("FLOATINGNUMBER")) {
               dtFloatingNumber = dt;
        } else
        if (name.equals("NOTANUMBER")) {
               dtNotANumber = dt;
        } else
        if (name.equals("ESTIMATED_MASS")) {
               dtEstimatedMass = dt;
        } else
        if (name.equals("WEIGHT")) {
               dtWeight = dt;
```

```
    }
}

// find the string-aggregate of enumeration values of NotaNumber
EEnumeration_type enu = (EEnumeration_type) dtNotANumber.getDomain(null);
A_string notaNumberStrings  = enu.getElements(null);

// create a repository
SdaiRepository rep = session.createRepository("Steel_warehouse", null);
rep.openRepository();

// create a model
SdaiModel mod = rep.createSdaiModel("example_3", exp1_schema);
mod.startReadWriteAccess();

// *********** creation of entity-instances **************
EDefined_type[] select = new EDefined_type[2];

EEntity ap1 = mod.createEntityInstance(steel_bar_def);
select[0] = dtFloatingNumber;
ap1.set(bar_length, 77.0, select);
select[0] = dtMeasuredMass;
ap1.set(bar_mass, 13.25, select);

EEntity ap2 = mod.createEntityInstance(steel_bar_def);
select[0] = dtNotANumber;
ap2.set(bar_length, ENotanumber.INDETERMINATE, select);
select[0] = dtEstimatedMass;
ap2.set(bar_mass, 10., select);

EEntity ap3 = mod.createEntityInstance(steel_bar_def);
select[0] = dtFloatingNumber;
ap3.set(bar_length, 77.0, select);
select[0] = dtComputedMass;
select[1] = dtFloatingNumber;
ap3.set(bar_mass, 14.7719, select);

// *********** searching for entity-instances **************
Aggregate insts = mod.getInstances(steel_bar_def);
SdaiIterator it = insts.createIterator();
while (it.next()) {
    EEntity sbar = (EEntity)insts.getCurrentMemberObject(it);
    double length, mass;
    int not_length, not_mass;

    String str_attributes = "Attributes of this steel bar: length = ";
    String value = "????";
    switch(sbar.testAttribute(bar_length, select)) {
    case 2: // int value
           if (select[0] == dtNotANumber) {
                  not_length = sbar.get_int(bar_length);
                  value = notaNumberStrings.getByIndex(not_length);
           }
           break;
```

```
            case 3: // double value
                  if (select[0] == dtFloatingNumber) {
                        length = sbar.get_double(bar_length);
                        value = String.valueOf(length);
                  }
                  break;
            default:
                  break;
            }
            str_attributes += value;

            str_attributes += ", mass = ";
            value = "????";
            switch(sbar.testAttribute(bar_mass, select)) {
            case 2: // int value
                  if (select[0] == dtComputedMass
                   && select[1] == dtNotANumber) {
                        not_length = sbar.get_int(bar_mass);
                        value = notaNumberStrings.getByIndex(not_length);
                  }
                  break;
            case 3: // double value
                  if (select[0] == dtMeasuredMass) {
                        length = sbar.get_double(bar_mass);
                        value = String.valueOf(length);
                  } else
                  if (select[0] == dtEstimatedMass) {
                        length = sbar.get_double(bar_mass);
                        value = String.valueOf(length);
                  } else
                  if (select[0] == dtComputedMass
                   && select[1] == dtFloatingNumber) {
                        length = sbar.get_double(bar_mass);
                        value = String.valueOf(length);
                  }
                  break;
            default:
                  break;
            }
            str_attributes += value;

            System.out.println(str_attributes);
         }
         session.closeSession();
      } // main
}
```

D.4 XList schema

```
SCHEMA xlist_schema;

ENTITY xx;
END_ENTITY;
```

```
TYPE xlist1 = LIST [0:?] OF xx;
END_TYPE;

TYPE xlist2 = LIST [0:?] OF xx;
END_TYPE;

TYPE xxlist = LIST [0:?] OF xlist1;
END_TYPE;

TYPE xselect = SELECT (xx, xlist1, xlist2);
END_TYPE;

ENTITY yy;
  attr1: xx;
  attr2: ARRAY [1:2] OF xx;
  attr3: xlist1;
  attr4: ARRAY [1:2] OF xlist1;
  attr5: xselect;
  attr6: ARRAY [1:2] OF xselect;
END_ENTITY;

END_SCHEMA;
```

D.4.1 SXlist_schema.java

Java class representing the EXPRESS schema Xlist_schema.

```
package SDAI.SXlist_schema;
import SDAI.lang.*;

public class SXlist_schema {
  ...
}
```

D.4.2 EXx.java

Java interface implementing the entity data type xx.

```
package SDAI.SXlist_schema;

public interface EXx extends SDAI.lang.EEntity {
}
```

D.4.3 CXx.java

Java class implementing the complex entity data type xx.

```
package SDAI.SXlist_schema;
import SDAI.lang.*;

public class CXx implements EXx extends ... {
    public static final SDAI.dictionary.EEntity_definition definition = ...;
```

```
    ...
}
```

D.4.4 AXx.java

Java class implementing aggregates of level 1 of entity xx.

```
package SDAI.SXlist_schema;
import SDAI.lang.*;

public class AXx extends AEntity {
    public EXx getByIndex(int index) throws SdaiException { ... }
    public EXx getCurrentMember(SdaiIterator iter) throws SdaiException { ... }
}
```

D.4.5 AaXx.java

Java class implementing aggregates of level 2 for entity xx.

```
package SDAI.SXlist_schema;
import SDAI.lang.*;

public class AaXx implements Aggregate extends ... {
    ...
    public boolean isMember(AXx value) throws SdaiException {…}
    public AXx getByIndex(int index) throws SdaiException {…}
    public AXx createAggregateByIndex(int index) throws SdaiException {…}
    public AXx addAggregateByIndex(int index) throws SdaiException {…}
    public AXx createAggregateUnordered() throws SdaiException {…}
    public void removeUnordered(AXx value) throws SdaiException {…}
    public AXx getCurrentMember(SdaiIterator iter) throws SdaiException {…}
    public AXx createAggregateAsCurrentMember(SdaiIterator iter) throws SdaiException {…}
    public AXx createAggregateBefore(SdaiIterator iter) throws SdaiException {…}
    public AXx createAggregateAfter(SdaiIterator iter) throws SdaiException {…}
}
```

D.4.6 EXlist1.java

Java interface representing the defined data type xlist1.

```
package SDAI.SXlist_schema;

public interface EXlist1 {
}
```

D.4.7 EXlist2.java

Java interface representing the defined data type xlist2.

```
package SDAI.SXlist_schema;
```

```
public interface EXlist2 {
}
```

D.4.8 EXxlist.java

Java interface representing the defined data type xxlist.

```
package SDAI.SXlist_schema;

public interface EXxlist {
}
```

D.4.9 AXselect.java

Java class implementing aggregates of level 1 for select data type xselect.

```
package SDAI.SXlist_schema;
import SDAI.lang.*;

public class AXselect implements Aggregate extends AEntity {
    ...
    public static final int sXlist1 = ...;
    public static final int sXlist2 = ...;

    public int testCurrentMember(SdaiIterator iter) throws SdaiException;
    public int testByIndex(int index) throws SdaiException;

    public boolean isMember(AXx value, EXlist1 node1)
        throws SdaiException { ... } // case sXlist1
    public boolean isMember(AXx value, EXlist2 node1)
        throws SdaiException { ... } // case sXlist2
    public AXx getByIndex(int index, EXlist1 node1)
        throws SdaiException { ... } // case sXlist1
    public AXx getByIndex(int index, EXlist2 node1)
        throws SdaiException { ... } // case sXlist2
    public AXx getCurrentMember(SdaiIterator iter, EXlist1 node1)
        throws SdaiException { ... } // case sXlist1
    public AXx getCurrentMember(SdaiIterator iter, EXlist2 node1)
        throws SdaiException { ... } // case sXlist2

    public AXx createAggregateByIndex(int index, EXlist1 node1)
        throws SdaiException { ... } // case sXlist1
    public AXx createAggregateByIndex(int index, EXlist2 node1)
        throws SdaiException { ... } // case sXlist2
    public AXx createAggregateAsCurrentMember (SdaiIterator iter, EXlist1 node1)
        throws SdaiException { ... } // case sXlist1
    public AXx createAggregateAsCurrentMember (SdaiIterator iter, EXlist2 node1)
        throws SdaiException { ... } // case sXlist2
    public AXx createAggregateBefore(SdaiIterator iter, EXlist1 node1)
        throws SdaiException { ... } // case sXlist1
    public AXx createAggregateBefore(SdaiIterator iter, EXlist2 node1)
        throws SdaiException { ... } // case sXlist2
    public AXx createAggregateAfter(SdaiIterator iter, EXlist1 node1)
```

```
        throws SdaiException { ... } // case sXlist1
    public AXx createAggregateAfter(SdaiIterator iter, EXlist2 node1)
        throws SdaiException { ... } // case sXlist2
    public AXx createAggregateUnordered(EXlist1 node1)
        throws SdaiException { ... } // case sXlist1
    public AXx createAggregateUnordered(EXlist2 node1)
        throws SdaiException { ... } // case sXlist2

    public void removeUnordered(AXx value, EXlist1 nod1)
        throws SdaiException { ... } // case sXlist1
    public void removeUnordered(AXx value, EXlist2 nod)
        throws SdaiException { ... } // case sXlist2
}
```

D.4.10 EYy.java

Java interface implementing the entity data type yy.

```
package SDAI.SXlist_schema;
import SDAI.lang.*;

public interface EYy extends EEntity {
    // methods for attribute: Attr1
    boolean testAttr1(EYy type) throws SdaiException;
    EXx getAttr1(EYy type) throws SdaiException;
    void setAttr1(EYy type, EXx value) throws SdaiException;
    void unsetAttr1(EYy type) throws SdaiException;

    // methods for attribute: Attr2
    boolean testAttr2(EYy type) throws SdaiException;
    AXx getAttr2(EYy type) throws SdaiException;
    AXx createAttr2(EYy type) throws SdaiException;
    void unsetAttr2(EYy type) throws SdaiException;

    // methods for attribute: Attr3
    boolean testAttr3(EYy type) throws SdaiException;
    AXx getAttr3(EYy type) throws SdaiException;
    AXx createAttr3(EYy type) throws SdaiException;
    void unsetAttr3(EYy type) throws SdaiException;

    // methods for attribute: Attr4
    boolean testAttr4(EYy type) throws SdaiException;
    AaXx getAttr4(EYy type) throws SdaiException;
    AaXx createAttr4(EYy type) throws SdaiException;
    void unsetAttr4(EYy type) throws SdaiException;

    // methods and constants for attribute: Attr5
    int sAttr5Xlist1 = ... ;
    int sAttr5Xlist2 = ... ;
    int testAttr5(EYy type) throws SdaiException; // returns 0, 1, ...
    EXx getAttr5(EYy type, EXx node1) throws SdaiException; // case 1
    AXx getAttr5(EYy type, EXlist1 node1)
        throws SdaiException; // case sAttr5Xlist1
```

```
    AXx getAttr5(EYy type, EXlist2 node1)
        throws SdaiException; // case sAttr5Xlist2
    void setAttr5(EYy type, EXx value)  throws SdaiException; // case 1
    AXx createAttr5(EYy type, EXlist1 node1)
        throws SdaiException; // case case sAttr5Xlist1
    AXx createAttr5(EYy type, EXlist2 node1)
        throws SdaiException; // case sAttr5Xlist2
    void unsetAttr5(EYy type) throws SdaiException;

    // methods for attribute: Attr6
    boolean testAttr6(EYy type) throws SdaiException;
    AXselect getAttr6(EYy type) throws SdaiException;
    AXselect createAttr6(EYy type) throws SdaiException;
    void unsetAttr6(EYy type) throws SdaiException;
}
```

D.4.11 CYy.java

Java class implementing the complex entity data type yy.

```
package SDAI.SXlist_schema;
import SDAI.lang.*;
import SDAI.dictionary.*;

public class CYy implements EYy extends ... {
    public static final SDAI.dictionary.EEntity_definition definition = ...;
    ...
    public static EAttribute attributeAttr1(EYy type) throws SdaiException { ... }
    public static EAttribute attributeAttr2(EYy type) throws SdaiException { ... }
    public static EAttribute attributeAttr3(EYy type) throws SdaiException { ... }
    public static EAttribute attributeAttr4(EYy type) throws SdaiException { ... }
    public static EAttribute attributeAttr5(EYy type) throws SdaiException { ... }
    public static EAttribute attributeAttr6(EYy type) throws SdaiException { ... }

    public static int usedInAttr1(EYy type, EXx instance, ASdaiModel domain, AEntity result )
      throws SdaiException { ... }
    public static int usedInAttr2(EYy type, EXx instance, ASdaiModel domain, AEntity result )
      throws SdaiException { ... }
    public static int usedInAttr3(EYy type, EXx instance, ASdaiModel domain, AEntity result )
      throws SdaiException { ... }
    public static int usedInAttr4(EYy type, EXx instance, ASdaiModel domain, AEntity result )
      throws SdaiException { ... }
    public static int usedInAttr5(EYy type, EXx instance, ASdaiModel domain, AEntity result )
      throws SdaiException { ... }
    public static int usedInAttr6(EYy type, EXx instance, ASdaiModel domain, AEntity result )
      throws SdaiException { ... }
}
```

D.4.12 AYy.java

Java class implementing aggregates of entity yy.

```
package SDAI.SXlist_schema;
```

```
import SDAI.lang.*;

public class AYy extends SDAI.lang.AEntity {
    public EYy getByIndex(int index) throws SdaiException {...}
    public EYy getCurrentMember(SdaiIterator iter) throws SdaiException {...}
}
```

D.5 XList schema - late binding access

Application with early binding access to the XList schema.
It is shown how to create simple aggregates, select of aggregates, aggregates of select for entity attributes.

File example_d5.java:

```
import SDAI.lang.*;
import SDAI.SXlist_schema.*;

public class XListLate {

    public static final void main(String argv[]) throws  SdaiException {
        SdaiSession session = SdaiSession.openSession();

        SdaiTransaction transaction = session.startTransactionReadWriteAccess();

        SdaiRepository rep = session.createRepository("exp_5_repo", null);
        rep.openRepository();


        // *********** creation of entity-instances **************
        SdaiModel mod = rep.createSdaiModel("exp_5_mod", SXlist_schema.class);
        mod.startReadWriteAccess();

        // creation of some xx instances
        EXx ex1 = (EXx) mod.createEntityInstance(CXx.class);
        EXx ex2 = (EXx) mod.createEntityInstance(CXx.class);
        EXx ex3 = (EXx) mod.createEntityInstance(CXx.class);
        EXx ex4 = (EXx) mod.createEntityInstance(CXx.class);
        EXx ex5 = (EXx) mod.createEntityInstance(CXx.class);

        // creation of a yy instance and setting the attributes
        EYy eyy = (EYy) mod.createEntityInstance(CYy.class);

        eyy.setAttr1(null, ex1);

        AXx ax2 = eyy.createAttr2(null);
        ax2.setByIndex(1, ex1);
        ax2.setByIndex(2, ex2);

        AXx ax3 = eyy.createAttr3(null);
        ax3.addByIndex(1, ex1);
        ax3.addByIndex(2, ex2);
```

```
        AaXx ax4 = eyy.createAttr4(null);
        AXx ax4_1 = ax4.createAggregateByIndex(1);
        ax4_1.addByIndex(1, ex1);
        ax4_1.addByIndex(2, ex2);
        AXx ax4_2 = ax4.createAggregateByIndex(2);
        ax4_1.addByIndex(1, ex1);
        ax4_1.addByIndex(2, ex2);

        eyy.setAttr5(null, ex1);
        // or
        AXx ax5_1 = eyy.createAttr5(null, (EXlist1)null);
        // or
        AXx ax5_2 = eyy.createAttr5(null, (EXlist2)null);

        AXselect as = eyy.createAttr6(null);
        as.setByIndex(3, ex1);
        AXx ax6_1 = as.createAggregateByIndex(1, (EXlist1)null);
        AXx ax6_2 = as.createAggregateByIndex(2, (EXlist2)null);

        // Printing something
        System.out.println(eyy.toString());
        System.out.println(ex1.toString());
        System.out.println(ex2.toString());
        System.out.println(ex3.toString());
        System.out.println(ex4.toString());
        System.out.println(ex5.toString());
        System.out.println(eyy.getAttr1(null).toString());
        System.out.println(eyy.getAttr2(null).toString());
        System.out.println(eyy.getAttr3(null).toString());
        System.out.println(eyy.getAttr4(null).toString());
        int result = eyy.testAttr5(null);
        switch (result) {
        case 1:
           System.out.println(eyy.getAttr5(null).toString());
           break;
        case eyy.sAttr5Xlist1:
           System.out.println(eyy.getAttr5(null, (EXlist1)null).toString());
           break;
        case eyy.sAttr5Xlist2:
           System.out.println(eyy.getAttr5(null, (EXlist2)null).toString());
           break;
        }
        System.out.println(eyy.getAttr6(null).toString());

        session.closeSession();
    } // main
}
```

Annex E
(informative)

Select path generator

```
/**
 * This SDAI application defines an algorithm to calculate and list the select paths
 * for the base_type of entity attributes and aggregate elements.
 * It is based on dictionary data as defined by the SDAI_dictionary_schema.
 * It accepts 3 command line parameters: schema name, named type name, optional attribute name
 * If the named type is an entity, then the third parameter - attribute name - is needed.
 * If the named type is a defined type, then the third parameter is absent.
*/
import java.util.*;
import SDAI.lang.*;
import SDAI.dictionary.*;

public class SelectPathGenerator {

    public static final void main(String argv[]) throws SdaiException {
        // check command line parameters
        if (argv.length < 2 || argv.length > 3) {
            System.out.println("\nSelect Path Generator, USAGE:"
            + "\njava SelectPathGenerator schema_name named_type [attribute_name]\n");
            return;
        }
        String schema_name = argv[0];
        String named_type_name = argv[1];
        String model_name = schema_name.toUpperPath() + "_DICTIONARY_DATA";

        SdaiSession session = SdaiSession.openSession();
        SdaiTransaction transaction = session.startTransactionReadWriteAccess();

        // find the schema_definition
        ESchema_definition schema_definition = null;
        SdaiModel dictionary_model = null;
        SchemaInstance si = session.getDataDictionary();
        ASdaiModel set_of_associated_models = si.getAssociatedModels();
        SdaiIterator it1 = set_of_associated_models.createIterator();
        while (it1.next()) {
            SdaiModel model = set_of_associated_models.getCurrentMember(it1);
            String m_name = model.getName();
            if (m_name.equals(model_name)) {
                dictionary_model = model;
                dictionary_model.startReadOnlyAccess();
                schema_definition = dictionary_model.getDefinedSchema();
                break;
            }
        }
        if (dictionary_model == null) {
            System.out.println("Dictionary data model for " + schema_name
            + " schema not found. Can not proceed");
```

```
    return;
}

// find the named_type
ENamed_type named_type = null;
ANamed_type aNamedTypes =
    (ANamed_type)dictionary_model.getInstances(ENamed_type.class);
aNamedTypes.attachIterator(it1);
while (it1.next()) {
    ENamed_type nt = aNamedTypes.getCurrentMember(it1);
    String name = nt.getName(null);
    if (name.equalsIgnoreCase(named_type_name)) {
        named_type = nt;
        break;
    }
}
if (named_type == null) {
    System.out.println("Named type " + named_type_name
    + " in schema " + schema_name + " not found");
    return;
}
if (named_type instanceof EEntity_definition) {
    if (argv.length != 3) {
        System.out.println("\nSelect Path Generator USAGE 1: "
        + "\n\njava SelectPathGenerator schema_name defined_type_name\n");
        System.out.println("\nSelect Path Generator USAGE 2: "
        + "\n\njava SelectPathGenerator schema_name entity_name attribute_name\n");
        return;
    }
    // find the attribute
    String attribute_name = argv[2];
    EEntity_definition entity_definition = (EEntity_definition)named_type;
    EExplicit_attribute attribute = null;
    AExplicit_attribute attributes = entity_definition.getExplicit_attributes(null);
    attributes.attachIterator(it1);
    while (it1.next()) {
        EExplicit_attribute att = attributes.getCurrentMember(it1);
        String name = att.getName(null);
        if (name.equalsIgnoreCase(attribute_name)) {
            attribute = att;
            break;
        }
    }
    if (attribute == null) {
        System.out.println("Attribute " + attribute_name + " of entity "
        + named_type_name + " in schema " + schema_name + " not found");
        return;
    }
    System.out.println("\nSelect paths for attribute " + attribute_name
    + " of entity " + named_type_name + " in schema " + schema_name + ":");
    EEntity base_type = attribute.getDomain(null);
    generateSelectPaths(base_type);

} else { // defined_type
```

```
            EDefined_type defined_type = (EDefined_type)named_type;
            System.out.println("\nSelect paths for defined type "
            + named_type_name + " in schema " + schema_name + ":");
            generateSelectPaths(defined_type);
        }
        session.closeSession();
    } // main

    /**
     * path 1 is reserved for entity values when additional defined_type parameters are not needed.
     * path 1 may be or not be present. Even if it is not present,
     * other paths are still numbered beginning with 2.
     * This algorithm determines if path 1 is present,
     * as well as calculates all the other select paths.
     */
    static void generateSelectPaths(EEntity base_type) throws SdaiException {
        Vector paths = new Vector();
        ADefined_type nodes = new ADefined_type(); // non-persistent list to keep the nodes
        EEntity ut = base_type;
        boolean select_present = false;
        while (ut instanceof EDefined_type) {
            EEntity domain = ((EDefined_type)ut).getDomain(null);
            if (domain instanceof ESelect_type) {
                boolean path_1_present = proceed((ESelect_type) domain, nodes, paths);
                removeDuplicatePaths(paths);
                printPaths(paths, path_1_present);
                select_present = true;
                break;
            }
            ut = domain;
        }
        if (ut instanceof EAggregation_type) {
            EEntity element = ((EAggregation_type)ut).getElement_type(null);
            for (;;) {
                boolean done_something = false;
                if (element instanceof EDefined_type) {
                    ut = element;
                    element = ((EDefined_type)ut).getDomain(null);
                    done_something = true;
                } else
                if (element instanceof EAggregation_type) {
                    ut = element;
                    element = ((EAggregation_type)ut).getElement_type(null);
                    done_something = true;
                }
                if (!done_something) {
                    ut = element;
                    break;
                }
            }
            if (ut instanceof ESelect_type) {
                boolean path_1_present = proceed((ESelect_type) ut, nodes, paths);
                removeDuplicatePaths(paths);
```

```
            printPaths(paths, path_1_present);
            select_present = true;
        }
    }
    if (!select_present) {
        System.out.println("\nThe base type of this attribute is not of " +
        + "select or aggregate of select type");
    }
}

static boolean proceed(ESelect_type st, ADefined_type nodes, Vector paths) throws SdaiException
{
    boolean path_1_present = false;
    ANamed_type ant = st.getSelections(null);
    SdaiIterator iant = ant.createIterator();
    boolean node_added = false;
    while (iant.next()) {
        EEntity entity = (ENamed_type)ant.getCurrentMemberObject(iant);
        while (entity instanceof EDefined_type) {
            EEntity domain = ((EDefined_type)entity).getDomain(null);
            if (!(domain instanceof ESelect_type)) {
                if (!node_added) {
                    nodes.addByIndex(nodes.getMemberCount()+1, (EDefined_type)entity);
                    node_added = true;
                }
            }
            entity = domain;
        }
        if (entity instanceof ESelect_type) {
            boolean path_1_inside = proceed((ESelect_type)entity, nodes, paths);
            if (!path_1_present) {
                path_1_present = path_1_inside;
            }
        }  else
        if (entity instanceof EEntity_definition && nodes.getMemberCount() == 0) {
            path_1_present = true;
        } else {
            SelectPath path = new SelectPath(nodes, entity);
            paths.addElement(path);
        }
        if (node_added){
            nodes.removeByIndex(nodes.getMemberCount());
            node_added = false;
        }
    }
    return path_1_present;
}

static void removeDuplicatePaths(Vector paths) {
    int path_count = paths.size();
    for (int i = 0; i < path_count - 1; i++) {
        SelectPath master = (SelectPath)paths.elementAt(i);
        for (int j = i + 1; j < path_count; j++) {
            SelectPath slave = (SelectPath)paths.elementAt(j);
```

```
                boolean identical = master.compare(slave);
                if (identical) {
                    paths.removeElementAt(j);
                    path_count--;
                    j--;
                }
            }
        }
    }

    static void printPaths(Vector paths, boolean print_path_1) throws SdaiException {
        System.out.println("");
        if (print_path_1) {
            System.out.println("1. ENTITY");
            System.out.println("\tEEntity");
        }
        for (int i = 0; i < paths.size(); i++) {
            SelectPath a_path = (SelectPath)paths.elementAt(i);
            a_path.printPath(i + 2);
        }
    }
}

class SelectPath {
    EDefined_type path[];
    EEntity value_type;

    SelectPath(ADefined_type nodes, EEntity entity) throws SdaiException  {
        int count = nodes.getMemberCount();
        path = new EDefined_type[count];
        for (int i = 1; i <= count; i++) {
            EDefined_type node = (EDefined_type)nodes.getByIndex(i);
            path[i-1] = node;
        }
        value_type = entity;
    }

    int size() {
        return path.length;
    }

    EDefined_type at(int i) {
        return path[i];
    }

    boolean compare(SelectPath other) {
        boolean identical;
        if (path.length != other.size()) {
            identical = false;
        } else {
            identical = true;
            for (int k = 0; k < path.length; k++) {
                EDefined_type dt1, dt2;
```

```
            if (path[k] != other.at(k)) {
                identical = false;
                break;
            }
        }
    }
    return identical;
}

void printPath(int nr) throws SdaiException {
    String value_type_name = null;
    String java_type_name = null;
    if (value_type instanceof EEntity_definition) {
        value_type_name = "ENTITY";
        java_type_name = "EEntity";
    }  else
    if (value_type instanceof EInteger_type) {
        value_type_name = "INTEGER";
        java_type_name = "int";
    }  else
    if (value_type instanceof ENumber_type) {
        value_type_name = "NUMBER";
        java_type_name = "double";
    }  else
    if (value_type instanceof EReal_type) {
        value_type_name = "REAL";
        java_type_name = "double";
    }  else
    if (value_type instanceof EString_type) {
        value_type_name = "STRING";
        java_type_name = "String";
    }  else
    if (value_type instanceof ELogical_type) {
        value_type_name = "LOGICAL";
        java_type_name = "int";
    }  else
    if (value_type instanceof EBoolean_type) {
        value_type_name = "BOOLEAN";
        java_type_name = "boolean";
    }  else
    if (value_type instanceof EBinary_type) {
        value_type_name = "BINARY";
        java_type_name = "Binary";
    } else
    if (value_type instanceof EEnumeration_type) {
        value_type_name = "ENUMERATION";
        java_type_name = "int";
    }  else
    if (value_type instanceof EAggregation_type) {
        value_type_name = "AGGREGATE";
        java_type_name = "Aggregate";
    }
    String coma = "";
    System.out.print(nr + ". " + value_type_name + ": ");
```

```
        for (int i = 0; i < path.length; i++) {
            System.out.print(coma + path[i].getName(null));
            coma = ", ";
        }
        System.out.println();
        coma = "";
        System.out.print("\t" + java_type_name + ": ");
        for (int i = 0; i < path.length; i++) {
            String node_name = path[i].getName(null);
            String normalized_node_name = "E" + node_name.substring(0,1).toUpperCase()
                + node_name.substring(1).toLowerCase();
            System.out.print(coma + normalized_node_name);
            coma = ", ";
        }
        System.out.println();
    }
}
```

Index

ICS 25.040.40

Price based on 100 pages