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

Part 22:

Implementation methods: Standard data
access interface

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

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



Contents	page
1 Scope	1
2 Normative references	2
3 Definitions and abbreviations	3
3.1 Terms defined in ISO 10303-1	3
3.2 Terms defined in ISO 10303-11	3
3.3 Other definitions	4
3.3.1 application schema	4
3.3.2 concurrent access	4
3.3.3 constraint	4
3.3.4 current schema	4
3.3.5 external schema	4
3.3.6 foreign schema	4
3.3.7 identifier	4
3.3.8 implementation class	4
3.3.9 iterator	4
3.3.10 native schema	4
3.3.11 repository	4
3.3.12 schema instance	4
3.3.13 SDAI-model	4
3.3.14 SDAI language binding	4
3.3.15 SDAI schema	5
3.3.16 session	5
3.3.17 validation	5
3.4 Abbreviations	5
4 SDAI overview	5
4.1 Data access interfaces	5
4.2 Operations and the session state	5
4.3 Repositories, schema instances, and SDAI-models	5
4.4 Transactions and access modes	6
4.5 The session, data dictionary and managing a population	7
4.6 SDAI parameter data schema	8
4.7 Functional specification	8
4.8 SDAI language bindings	9
4.9 Error handling	10
5 Fundamental principles	10

© ISO 1998

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

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

Printed in Switzerland

6 SDAI dictionary schema	11
6.1 Introduction	11
6.2 Fundamental concepts and assumptions	12
6.3 SDAI dictionary schema type definitions	12
6.3.1 base_type	12
6.3.2 constructed_type	12
6.3.3 underlying_type	13
6.3.4 type_or_rule	13
6.3.5 explicit_or_derived	13
6.3.6 express_id	13
6.3.7 info_object_id	14
6.4 SDAI dictionary schema entity definitions	14
6.4.1 schema_definition	14
6.4.2 interface_specification	15
6.4.3 interfaced_item	15
6.4.4 explicit_item_id	16
6.4.5 used_item	16
6.4.6 referenced_item	16
6.4.7 implicit_item_id	17
6.4.8 external_schema	17
6.4.9 domain_equivalent_type	18
6.4.10 named_type	18
6.4.11 defined_type	19
6.4.12 entity_definition	19
6.4.13 attribute	20
6.4.14 derived_attribute	21
6.4.15 explicit_attribute	21
6.4.16 inverse_attribute	22
6.4.17 uniqueness_rule	22
6.4.18 where_rule	23
6.4.19 global_rule	23
6.4.20 simple_type	24
6.4.21 number_type	24
6.4.22 integer_type	24
6.4.23 real_type	25
6.4.24 string_type	25
6.4.25 binary_type	26
6.4.26 logical_type	26
6.4.27 boolean_type	26
6.4.28 enumeration_type	27
6.4.29 select_type	27
6.4.30 aggregation_type	27
6.4.31 variable_size_aggregation_type	28
6.4.32 set_type	28
6.4.33 bag_type	28
6.4.34 list_type	29
6.4.35 array_type	29
6.4.36 bound	30
6.4.37 population_dependent_bound	30

6.4.38 integer_bound	30
7 SDAI session schema	31
7.1 Introduction	31
7.2 Fundamental concepts and assumptions	31
7.3 SDAI session schema type definitions	32
7.3.1 access_type	32
7.3.2 error_base	32
7.3.3 time_stamp	32
7.4 SDAI session schema entity definitions	33
7.4.1 sdai_session	33
7.4.2 implementation	34
7.4.3 sdai_repository	35
7.4.4 sdai_repository_contents	36
7.4.5 sdai_transaction	36
7.4.6 event	36
7.4.7 error_event	37
8 SDAI population schema	38
8.1 Introduction	38
8.2 Fundamental concepts and assumptions	39
8.3 SDAI population schema type definitions	39
8.3.1 schema_definition	39
8.3.2 entity_definition	39
8.4 SDAI population schema entity definitions	39
8.4.1 schema_instance	39
8.4.2 sdai_model	41
8.4.3 sdai_model_contents	42
8.4.4 entity_extent	42
8.4.5 scope	43
9 SDAI parameter data schema	44
9.1 Introduction	44
9.2 Fundamental concepts and assumptions	45
9.3 SDAI parameter data schema type definitions	45
9.3.1 primitive	45
9.3.2 assignable_primitive	46
9.3.3 aggregate_primitive	46
9.3.4 string_value	46
9.3.5 binary_value	47
9.3.6 integer_value	47
9.3.7 real_value	47
9.3.8 number_value	47
9.3.9 boolean_value	47
9.3.10 logical_value	48
9.3.11 bound_instance_value	48
9.3.12 query_source	48
9.4 SDAI parameter data schema entity definitions	49
9.4.1 iterator	49

9.4.2 entity_instance	49
9.4.3 application_instance	50
9.4.4 sdai_instance	50
9.4.5 dictionary_instance	51
9.4.6 session_instance	51
9.4.7 attribute_value	51
9.4.8 select_value	52
9.4.9 select_aggregate_instance	53
9.4.10 enumeration_value	53
9.4.11 aggregate_instance	54
9.4.12 unordered_collection	54
9.4.13 set_instance	54
9.4.14 bag_instance	55
9.4.15 ordered_collection	55
9.4.16 list_instance	56
9.4.17 schema_defined_list_instance	56
9.4.18 non_persistent_list_instance	56
9.4.19 array_instance	57
9.4.20 application_indexed_array_instance	57
10 SDAI operations	58
10.1 Introduction	58
10.2 Fundamental concepts and assumptions	59
10.3 Environment operations	60
10.3.1 Open session	60
10.4 Session operations	61
10.4.1 Record error	61
10.4.2 Start event recording	61
10.4.3 Stop event recording	62
10.4.4 Close session	63
10.4.5 Open repository	63
10.4.6 Start transaction read-write access	64
10.4.7 Start transaction read-only access	64
10.4.8 Commit	65
10.4.9 Abort	66
10.4.10 End transaction access and commit	68
10.4.11 End transaction access and abort	68
10.4.12 Create non-persistent list	69
10.4.13 Delete non-persistent list	69
10.4.14 SDAI query	70
10.5 Repository operations	72
10.5.1 Create SDAI-model	72
10.5.2 Create schema instance	73
10.5.3 Close repository	75
10.6 Schema instance operations	75
10.6.1 Delete schema instance	75
10.6.2 Rename schema instance	76
10.6.3 Add SDAI-model	77
10.6.4 Remove SDAI-model	77

10.6.5 Validate global rule	78
10.6.6 Validate uniqueness rule	79
10.6.7 Validate instance reference domain	80
10.6.8 Validate schema instance	81
10.6.9 Is validation current	82
10.7 SDAI-model operations	82
10.7.1 Delete SDAI-model	82
10.7.2 Rename SDAI-model	83
10.7.3 Start read-only access	84
10.7.4 Promote SDAI-model to read-write	84
10.7.5 End read-only access	85
10.7.6 Start read-write access	85
10.7.7 End read-write access	86
10.7.8 Get entity definition	87
10.7.9 Create entity instance	87
10.7.10 Undo changes	88
10.7.11 Save changes	89
10.8 Scope operations	90
10.8.1 Add to scope	90
10.8.2 Is scope owner	91
10.8.3 Get scope	91
10.8.4 Remove from scope	92
10.8.5 Add to export list	93
10.8.6 Remove from export list	93
10.8.7 Scoped delete	94
10.8.8 Scoped copy	95
10.8.9 Validate scope reference restrictions	96
10.9 Type operations	96
10.9.1 Get complex entity definition	96
10.9.2 Is subtype of	97
10.9.3 Is SDAI subtype of	98
10.9.4 Is domain equivalent with	98
10.10 Entity instance operations	99
10.10.1 Get attribute	99
10.10.2 Test attribute	100
10.10.3 Find entity instance SDAI-model	100
10.10.4 Get instance type	101
10.10.5 Is instance of	101
10.10.6 Is kind of	102
10.10.7 Is SDAI kind of	103
10.10.8 Find entity instance users	103
10.10.9 Find entity instance used in	104
10.10.10 Get attribute value bound	105
10.10.11 Find instance roles	106
10.10.12 Find instance data types	106
10.11 Application instance operations	107
10.11.1 Copy application instance	107
10.11.2 Delete application instance	108
10.11.3 Put attribute	109

10.11.4 Unset attribute value	110
10.11.5 Create aggregate instance	110
10.11.6 Get persistent label	111
10.11.7 Get session identifier	112
10.11.8 Get description	113
10.11.9 Validate where rule	113
10.11.10 Validate required explicit attributes assigned	114
10.11.11 Validate inverse attributes	114
10.11.12 Validate explicit attributes references	115
10.11.13 Validate aggregates size	116
10.11.14 Validate aggregates uniqueness	117
10.11.15 Validate array not optional	117
10.11.16 Validate string width	118
10.11.17 Validate binary width	119
10.11.18 Validate real precision	120
10.12 Entity instance aggregate operations	121
10.12.1 Get member count	121
10.12.2 Is member	121
10.12.3 Create iterator	122
10.12.4 Delete iterator	122
10.12.5 Beginning	123
10.12.6 Next	123
10.12.7 Get current member	124
10.12.8 Get value bound by iterator	125
10.12.9 Get lower bound	125
10.12.10 Get upper bound	126
10.13 Application instance aggregate operations	127
10.13.1 Create aggregate instance as current member	127
10.13.2 Put current member	128
10.13.3 Remove current member	128
10.14 Application instance unordered collection operations	129
10.14.1 Add unordered	129
10.14.2 Create aggregate instance unordered	130
10.14.3 Remove unordered	131
10.15 Entity instance ordered collection operations	131
10.15.1 Get by index	131
10.15.2 End	132
10.15.3 Previous	132
10.15.4 Get value bound by index	133
10.16 Application instance ordered collection operations	134
10.16.1 Put by index	134
10.16.2 Create aggregate instance by index	135
10.17 Entity instance array operations	136
10.17.1 Test by index	136
10.17.2 Test current member	136
10.17.3 Get lower index	137
10.17.4 Get upper index	137
10.18 Application instance array operations	138
10.18.1 Unset value by index	138

10.18.2 Unset value current member	139
10.18.3 Reindex array	139
10.18.4 Reset array index	140
10.19 Application instance list operations	141
10.19.1 Add before current member	141
10.19.2 Add after current member	141
10.19.3 Add by index	142
10.19.4 Create aggregate instance before current member	143
10.19.5 Create aggregate instance after current member	144
10.19.6 Add aggregate instance by index	145
10.19.7 Remove by index	146
11 SDAI errors	146
12 SDAI state model	150
12.1 State model for transaction level 1	154
12.1.1 No Session 1 State	155
12.1.2 Session 1 State	155
12.1.3 Repository Open 1 State	155
12.1.4 SDAI-Model Started RO 1 State	155
12.1.5 SDAI-Model Started RW 1 State	155
12.1.6 State transitions	155
12.2 State model for transaction level 2	156
12.2.1 No Session 2 State	156
12.2.2 Session 2 State	156
12.2.3 Repository Open 2 State	156
12.2.4 SDAI-Model Started RO 2 State	157
12.2.5 SDAI-Model Started RW 2 State	157
12.2.6 State transitions	157
12.3 State model for transaction level 3	158
12.3.1 No Session 3 State	158
12.3.2 Session 3 State	158
12.3.3 Transaction Started RO 3 State	158
12.3.4 Transaction Started RW 3 State	158
12.3.5 Repository Open 3 State	158
12.3.6 RO Repository Open 3 State	158
12.3.7 RW Repository Open 3 State	159
12.3.8 RO Model Started RO 3 State	159
12.3.9 RW Model Started RO 3 State	159
12.3.10 RW Model Started RW 3 State	159
12.3.11 State transitions	159
13 Implementation classes	160
13.1 Implementations of SDAI	160
13.1.1 Levels of transaction	160
13.1.2 Levels of expression evaluation for validation and derived attributes	161
13.1.3 Levels of session event recording support	161
13.1.4 Levels of scope support	162
13.1.5 Levels of domain equivalence support	162

13.2 Implementations class specification	162
13.2.1 Implementation class 1	162
13.2.2 Implementation class 2	163
13.2.3 Implementation class 3	163
13.2.4 Implementation class 4	163
13.2.5 Implementation class 5	163
13.2.6 Implementation class 6	164
13.2.7 Implementation class 7	164
13.3 Operations required by implementations class	164

Annexes

A Mapping EXPRESS into SDAI dictionary schema constructs	168
A.1 EXPRESS Language Constructs	168
A.1.1 Interface specification	168
A.1.2 EXPRESS ABSTRACT	168
A.1.3 Interpretation of the EXPRESS SUPERTYPE expression - AND and ANDOR ...	169
A.1.4 Bounds and bound expressions	170
A.1.5 Attribute redeclaration	171
A.2 Domain equivalence information	171
A.2.1 Dictionary constructs	171
A.2.2 Algorithms and methods for declaring domain equivalence	171
B Protocol Implementation Conformance Statement (PICS) proforma	174
C Information object registration	176
C.1 Document identification	176
C.2 Schema identification	176
D EXPRESS-G diagrams	178
E SDAI schema EXPRESS listing	188
Index	189

Figures	page
Figure 1 - An example SDAI storage structure	6
Figure 2 - The SDAI data architecture element relationships	7
Figure D.1 - SDAI dictionary schema EXPRESS-G diagram 1 of 5	178
Figure D.2 - SDAI dictionary schema EXPRESS-G diagram 2 of 5	179
Figure D.3 - SDAI dictionary schema EXPRESS-G diagram 3 of 5	180
Figure D.4 - SDAI dictionary schema EXPRESS-G diagram 4 of 5	181
Figure D.5 - SDAI dictionary schema EXPRESS-G diagram 5 of 5	182
Figure D.6 - SDAI session schema EXPRESS-G diagram 1 of 2	183
Figure D.7 - SDAI session schema EXPRESS-G diagram 2 of 2	184
Figure D.8 - SDAI population schema EXPRESS-G diagram	185

Figure D.9 - SDAI parameter data schema EXPRESS-G diagram 1 of 2 186

Figure D.10 - SDAI parameter data schema EXPRESS-G diagram 2 of 2 187

Tables	page
Table - 1 Attribute representations supported in query	71
Table - 2 SDAI error indicators	148
Table - 3 SDAI operations groupings	151
Table - 4 State transitions for transaction level 1	156
Table - 5 State transitions for transaction level 2	157
Table - 6 State transitions for transaction level 3	160
Table - 7 Operations required by implementation class	165

Foreword

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

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

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

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

- Part 1, Overview and fundamental principles;
- Part 11, Description method: EXPRESS language reference manual;
- Part 12, Description method: EXPRESS-I language reference manual;
- Part 21, Implementation method: Clear text encoding of the exchange structure;
- Part 22, Implementation method: Standard data access interface specification;
- Part 23, Implementation method: C++ language binding to the standard data access interface;
- Part 24, Implementation method: C language binding to the standard data access interface;
- Part 26, Implementation method: Interface definition language binding to the standard data access;
- Part 31, Conformance testing methodology and framework: General concepts;
- Part 32, Conformance testing methodology and framework: Requirements on testing laboratories and clients;
- Part 33, Conformance testing methodology and framework: Structure and use of abstract test suites;
- Part 34, Conformance testing methodology and framework: Abstract test methods;
- Part 35, Conformance testing methodology and framework: Abstract test methods for SDAI implementations;
- Part 41, Integrated generic resource: Fundamentals of product description and support;

- Part 42, Integrated generic resource: Geometric and topological representation;
- Part 43, Integrated generic resource: Representation structures;
- Part 44, Integrated generic resource: Product structure configuration;
- Part 45, Integrated generic resource: Materials;
- Part 46, Integrated generic resource: Visual presentation;
- Part 47, Integrated generic resource: Shape variation tolerances;
- Part 49, Integrated generic resource: Process structure and properties;
- Part 101, Integrated application resource: Draughting;
- Part 104, Integrated application resource: Finite element analysis;
- Part 105, Integrated application resource: Kinematics;
- Part 106, Integrated application resource: Building construction core model;
- Part 201, Application protocol: Explicit draughting;
- Part 202, Application protocol: Associative draughting;
- Part 203, Application protocol: Configuration controlled design;
- Part 204, Application protocol: Mechanical design using boundary representation;
- Part 205, Application protocol: Mechanical design using surface representation;
- Part 207, Application protocol: Sheet metal die planning and design;
- Part 208, Application protocol: Life cycle management - Change process;
- Part 209, Application protocol: Composite and metallic structural analysis and related design;
- Part 210, Application protocol: Electronic assembly, interconnect, and packaging design;
- Part 212, Application protocol: Electrotechnical design and installation;
- Part 213, Application protocol: Numerical control process plans for machined parts;
- Part 214, Application protocol: Core data for automotive design;
- Part 215, Application protocol: Ship arrangement;
- Part 216, Application protocol: Ship moulded forms;

- Part 217, Application protocol: Ship piping;
- Part 218, Application protocol: Ship structures;
- Part 220, Application protocol: Process planning, manufacture, and assembly of layered electronic products;
- Part 221, Application protocol: Functional data and their schematic representation for process plant;
- Part 222, Application protocol: Exchange of product data for composite structures;
- Part 223, Application protocol: Exchange of design and manufacturing product information for cast parts;
- Part 224, Application protocol: Mechanical product definition for process plans using machining features;
- Part 225, Application protocol: Building elements using explicit shape representation;
- Part 226, Application protocol: Ship mechanical systems;
- Part 227, Application protocol: Plant spatial configuration;
- Part 228, Application protocol: Building services: Heating, ventilation, and air conditioning;
- Part 229, Application protocol: Exchange of design and manufacturing product information for forged parts;
- Part 230, Application protocol: Building structural frame: Steelwork;
- Part 231, Application protocol: Process engineering data: Process design and process specification of major equipment;
- Part 232, Application protocol: Technical data packaging core information and exchange;
- Part 301, Abstract test suite: Explicit draughting;
- Part 302, Abstract test suite: Associative draughting;
- Part 303, Abstract test suite: Configuration controlled design;
- Part 304, Abstract test suite: Mechanical design using boundary representation;
- Part 305, Abstract test suite: Mechanical design using surface representation;
- Part 307, Abstract test suite: Sheet metal die planning and design;
- Part 308, Abstract test suite: Life cycle management - Change process;

- Part 309, Abstract test suite: Composite and metallic structural analysis and related design;
- Part 310, Abstract test suite: Electronic assembly, interconnect, and packaging design;
- Part 312, Abstract test suite: Electrotechnical design and installation;
- Part 313, Abstract test suite: Numerical control process plans for machined parts;
- Part 314, Abstract test suite: Core data for automotive mechanical design processes;
- Part 315, Abstract test suite: Ship arrangement;
- Part 316, Abstract test suite: Ship moulded forms;
- Part 317, Abstract test suite: Ship piping;
- Part 318, Abstract test suite: Ship structures;
- Part 320, Abstract test suite: Process planning, manufacture, and assembly of layered electronic products;
- Part 321, Abstract test suite: Functional data and their schematic representation for process plant;
- Part 322, Abstract test suite: Exchange of product data for composite structures;
- Part 323, Abstract test suite: Exchange of design and manufacturing product information for cast parts;
- Part 324, Abstract test suite: Mechanical product definition for process plans using machining features;
- Part 325, Abstract test suite: Building elements using explicit shape representation;
- Part 326, Abstract test suite: Ship mechanical systems;
- Part 327, Abstract test suite: Plant spatial configuration;
- Part 328, Abstract test suite: Building services: Heating, ventilation, and air conditioning;
- Part 329, Abstract test suite: Exchange of design and manufacturing product information for forged parts;
- Part 330, Abstract test suite: Building structural frame: Steelwork;
- Part 331, Abstract test suite: Process engineering data: Process design and process specification of major equipment;
- Part 332, Abstract test suite: Technical data packaging core information and exchange;

- Part 501, Application interpreted construct: Edge-based wireframe;
- Part 502, Application interpreted construct: Shell-based wireframe;
- Part 503, Application interpreted construct: Geometrically bounded 2D wireframe;
- Part 504, Application interpreted construct: Draughting annotation;
- Part 505, Application interpreted construct: Drawing structure and administration;
- Part 506, Application interpreted construct: Draughting elements;
- Part 507, Application interpreted construct: Geometrically bounded surface;
- Part 508, Application interpreted construct: Non-manifold surface;
- Part 509, Application interpreted construct: Manifold surface;
- Part 510, Application interpreted construct: Geometrically bounded wireframe;
- Part 511, Application interpreted construct: Topologically bounded surface;
- Part 512, Application interpreted construct: Faceted boundary representation;
- Part 513, Application interpreted construct: Elementary boundary representation;
- Part 514, Application interpreted construct: Advanced boundary representation;
- Part 515, Application interpreted construct: Constructive solid geometry;
- Part 517, Application interpreted construct: Mechanical design geometric presentation;
- Part 518, Application interpreted construct: Mechanical design shaded representation.

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

- Parts 11 to 13 specify the description methods,
- Parts 21 to 26 specify the implementation methods,
- Parts 31 to 35 specify the conformance testing methodology and framework,
- Parts 41 to 49 specify the integrated generic resources,
- Parts 101 to 106 specify the integrated application resources,
- Parts 201 to 232 specify the application protocols,

- Parts 301 to 332 specify the abstract test suites, and
- Parts 501 to 518 specify the application interpreted constructs.

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

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

Introduction

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

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

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

Major subdivisions in this part of ISO 10303 are:

- SDAI environment constructs defined using EXPRESS found in clauses 6 through 9;
- SDAI operations, errors and state found in clauses 10 through 12;
- Implementation classes of the SDAI functionality to which implementations claim conformance found in clause 13.

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

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

This page is intentionally blank.

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

1 Scope

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

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

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

- access to and manipulation of instances of entities described using the EXPRESS data specification language;
- access to multiple data repositories by a single application at the same time;
- capabilities for an application to organize operations into groups whose effect can be saved or cancelled at the discretion of the application;
- access to a dictionary describing the data elements that can be manipulated by an application;
- ability to invoke the validation of the constraints specified using EXPRESS at the discretion of the application;
- support for managing the dependency relationships between entity instances;
- capabilities to describe logical collections of entity instances that define the population over which entity instance to entity instance references are allowed;
- capabilities to describe logical collections of entity instances that define the population over which global rules are validated;
- support for the use of data created within the context of one schema in the context of another schema.

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

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

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

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

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

- data access and manipulation operations that are specific to the semantics of the data;
- specification of the mechanisms or formats by which data is held in a data repository;
- creation, deletion, and naming of the data repositories available via the SDAI.

2 Normative references

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

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

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

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

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

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

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

3 Definitions and abbreviations

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

3.1 Terms defined in ISO 10303-1

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

- application;
- application protocol (AP);
- data;
- implementation method;
- information;
- information model;
- product information model;
- protocol implementation conformance statement (PICS);
- structure.

3.2 Terms defined in ISO 10303-11

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

- complex entity data type;
- data type;
- entity;
- entity data type;
- entity instance;
- instance;
- population;
- value.

3.3 Other definitions

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

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

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

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

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

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

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

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

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

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

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

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

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

3.3.11 repository: an identifiable data storage facility.

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

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

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

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

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

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

3.4 Abbreviations

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

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

4 SDAI overview

4.1 Data access interfaces

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

4.2 Operations and the session state

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

4.3 Repositories, schema instances, and SDAI-models

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

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

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

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

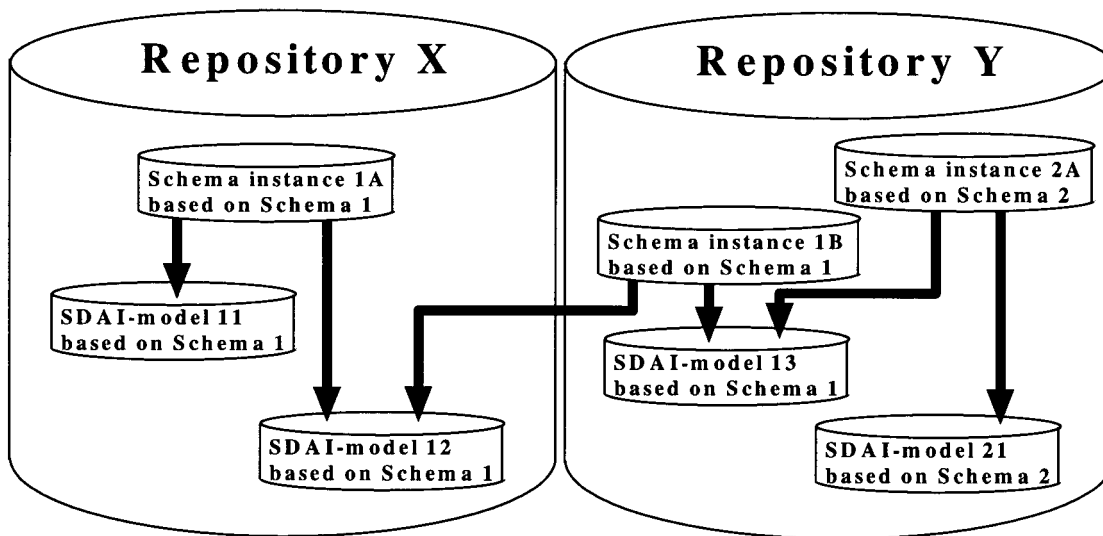


Figure 1 - An example SDAI storage structure

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

4.4 Transactions and access modes

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

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

4.5 The session, data dictionary and managing a population

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

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

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

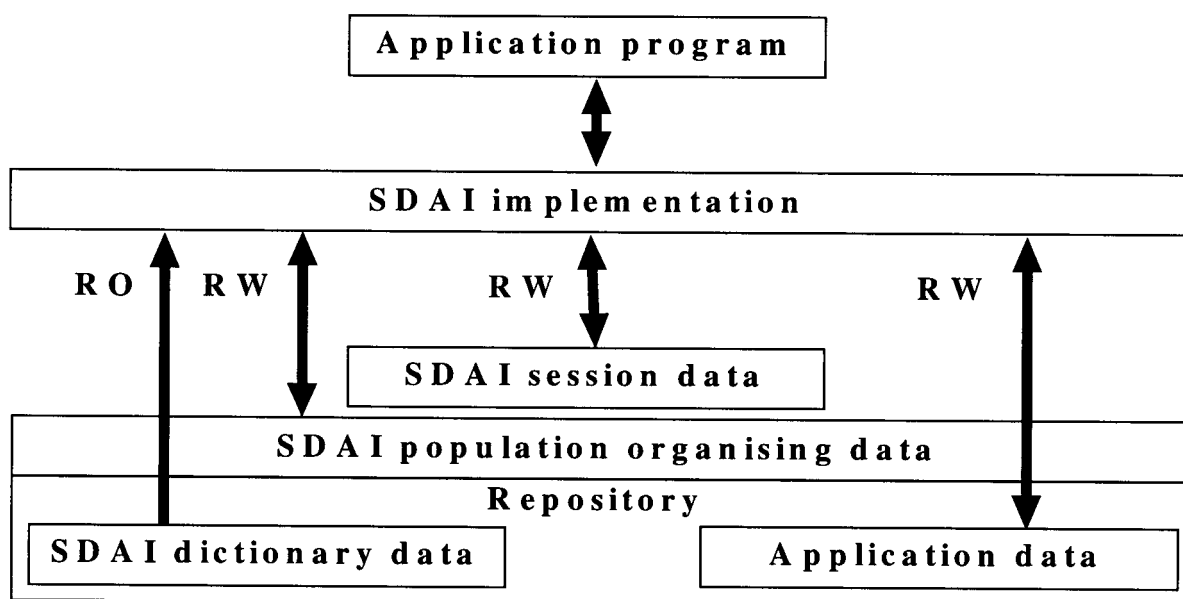


Figure 2 - The SDAI data architecture element relationships

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

4.6 SDAI parameter data schema

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

4.7 Functional specification

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

- environment operations that establish an SDAI session (see 10.3);
 - session operations that allow an application to manage the transactions, repositories, and queries in the session (see 10.4);
 - repository operations that allow an application to manage access to SDAI-models within the repositories (see 10.5);
 - schema instance operations that allow an application to manage the association of SDAI-models with schema instances and validate EXPRESS global rules and references within a schema instance (see 10.6);
 - SDAI-model operations that allow an application to create instances and manage the SDAI-model access (see 10.7);
 - operations that allow an application to create and manage support for the dependency relationships between entity instances (see 10.8);
- NOTE - These dependency relationships are based upon the SCOPE construct described in ISO 10303-21.
- operations that allow an application to examine type information and domain equivalency information (see 10.9);
 - entity instance operations that allow an application to manipulate instances of entity data types found in both SDAI-defined schemas and application schemas (see 10.10);

- application instance operations that allow an application to create, modify, delete and validate instances of entity data types and create instances of aggregates defined in application schemas (see 10.11);
- categories of entity and application instance operations that allow an application to create, delete, manipulate and modify instances of various aggregate types (see 10.12 through 10.19).

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

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

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

4.8 SDAI language bindings

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

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

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

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

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

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

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

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

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

EXAMPLE 4 - An operation to set the value of attribute "a1" of entity instance "i1" of entity type "t1" might be written in an early binding as `SetValuel1a1( i1, value )`.

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

4.9 Error handling

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

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

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

5 Fundamental principles

The following concepts and assumptions apply:

- During the course of an SDAI session, a unique and unchanging identifier is available for each instance of an entity or aggregation described in an SDAI or application schema;
- The EXPRESS schema upon which SDAI-models and schema instances are based is the completely expanded form of a schema in that all items interfaced from other schemas are resolved into the referencing schema making the schema complete with no references to other schemas (see A.1);

- The process that makes the population of the SDAI dictionary schema available in a session expands any complex entity data types that result from either an explicit or implicit ANDOR or AND constraint following the approach given in A.1.3;
- When required by the implementation class, SDAI implementations make available population of the SDAI dictionary schema with the EXPRESS items specified in the SDAI session schema and application schemas;
- The schemas defined in clauses 6 through 9 are not application protocols or product information models and as such take efficiency, convenience and the SDAI operations into account;
- Entity instances based upon one schema may be shared in the context of other schemas based upon domain equivalence (see A.2) and the domain equivalence constructs defined in the SDAI dictionary schema (see 6.4.1, 6.4.8 and 6.4.9);
- EXPRESS constraints are validated only at the request of the application. When validation occurs, all creation, modification and deletion of SDAI-models, schema instances, SCOPE constructs and application instances is taken into account.

6 SDAI dictionary schema

The following EXPRESS declaration begins the SDAI dictionary schema.

EXPRESS specification:

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

6.1 Introduction

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

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

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

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

6.2 Fundamental concepts and assumptions

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

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

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

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

6.3 SDAI dictionary schema type definitions

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

6.3.1 base_type

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

EXPRESS specification:

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

6.3.2 constructed_type

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

EXPRESS specification:

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

(*

6.3.3 underlying_type

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

EXPRESS specification:

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

6.3.4 type_or_rule

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

EXPRESS specification:

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

6.3.5 explicit_or_derived

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

EXPRESS specification:

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

6.3.6 express_id

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

EXPRESS specification:

```

*)
TYPE express_id = STRING;
END_TYPE;
( *

```

6.3.7 info_object_id

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

EXPRESS specification:

```

*)
TYPE info_object_id = STRING;
END_TYPE;
( *

```

6.4 SDAI dictionary schema entity definitions

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

6.4.1 schema_definition

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

EXPRESS specification:

```

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

```

Attribute definitions:

name: the name of the schema.

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

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

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

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

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

Formal propositions:

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

6.4.2 interface_specification

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

EXPRESS specification:

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

Attribute definitions:

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

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

implicit_items: the items implicitly interfaced.

6.4.3 interfaced_item

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

EXPRESS specification:

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

```

    foreign_schema_id : express_id;
END_ENTITY;
(*

```

Attribute definitions:

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

6.4.4 explicit_item_id

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

EXPRESS specification:

```

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

```

Attribute definitions:

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

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

6.4.5 used_item

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

EXPRESS specification:

```

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

```

6.4.6 referenced_item

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

EXPRESS specification:

```

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

```

6.4.7 implicit_item_id

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

EXPRESS specification:

```

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

```

Attribute definitions:

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

6.4.8 external_schema

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

EXPRESS specification:

```

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

```

Attribute definitions:

name: the name of the external schema.

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

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

Informal propositions:

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

6.4.9 domain_equivalent_type

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

EXPRESS specification:

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

Attribute definitions:

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

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

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

Informal propositions:

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

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

6.4.10 named_type

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

EXPRESS specification:

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

```
END_ENTITY;
( *
```

Attribute definitions:

name: the name of the data type.

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

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

6.4.11 defined_type

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

EXPRESS specification:

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

Attribute definitions:

domain: the underlying type of the defined type.

6.4.12 entity_definition

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

EXPRESS specification:

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

Attribute definitions:

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

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

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

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

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

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

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

6.4.13 attribute

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

EXPRESS specification:

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

Attribute definitions:

name: the name of the attribute.

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

6.4.14 derived_attribute

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

EXPRESS specification:

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

Attribute definitions:

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

redeclaring: if present, the attribute being redeclared.

6.4.15 explicit_attribute

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

EXPRESS specification:

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

Attribute definitions:

domain: the data type referenced by the attribute.

redeclaring: if present, the attribute being redeclared.

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

6.4.16 inverse_attribute

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

EXPRESS specification:

```

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

Attribute definitions:

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

redeclaring: if present, the attribute being redeclared.

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

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

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

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

6.4.17 uniqueness_rule

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

EXPRESS specification:

```

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

```

Attribute definitions:

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

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

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

6.4.18 where_rule

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

EXPRESS specification:

```

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

```

Attribute definitions:

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

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

6.4.19 global_rule

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

EXPRESS specification:

```

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

```

```

    parent_schema : schema_definition;
END_ENTITY;
(*

```

Attribute definitions:

name: the name of the rule.

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

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

parent_schema: the schema in which the rule is declared.

6.4.20 simple_type

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

EXPRESS specification:

```

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

```

6.4.21 number_type

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

EXPRESS specification:

```

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

```

6.4.22 integer_type

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

EXPRESS specification:

```

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

```

6.4.23 real_type

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

EXPRESS specification:

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

Attribute definitions:

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

Informal propositions:

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

6.4.24 string_type

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

EXPRESS specification:

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

Attribute definitions:

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

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

Informal propositions:

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

6.4.25 binary_type

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

EXPRESS specification:

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

Attribute definitions:

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

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

Informal propositions:

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

6.4.26 logical_type

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

EXPRESS specification:

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

6.4.27 boolean_type

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

EXPRESS specification:

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

6.4.28 enumeration_type

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

EXPRESS specification:

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

Attribute definitions:

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

6.4.29 select_type

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

EXPRESS specification:

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

Attribute definitions:

selections: the set of selectable types.

6.4.30 aggregation_type

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

EXPRESS specification:

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

Attribute definitions:

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

6.4.31 variable_size_aggregation_type

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

EXPRESS specification:

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

Attribute definitions:

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

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

Informal propositions:

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

6.4.32 set_type

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

EXPRESS specification:

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

6.4.33 bag_type

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

EXPRESS specification:

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

6.4.34 list_type

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

EXPRESS specification:

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

Attribute definitions:

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

6.4.35 array_type

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

EXPRESS specification:

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

Attribute definitions:

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

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

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

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

Informal propositions:

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

6.4.36 bound

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

EXPRESS specification:

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

6.4.37 population_dependent_bound

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

EXPRESS specification:

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

6.4.38 integer_bound

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

EXPRESS specification:

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

Attribute definitions:

bound_value: the integer value for the bound.

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

7 SDAI session schema

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

EXPRESS specification:

```
*)
SCHEMA SDAI_session_schema;

REFERENCE FROM SDAI_parameter_data_schema
  ( entity_instance,
    aggregate_instance );

USE FROM SDAI_population_schema;
(*
```

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

SDAI_parameter_data_schema	clause 9
SDAI_population_schema	clause 8

7.1 Introduction

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

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

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

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

7.2 Fundamental concepts and assumptions

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

7.3 SDAI session schema type definitions

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

7.3.1 access_type

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

EXPRESS specification:

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

Enumeration items:

read_only: the value indicating RO access is available.

read_write: the value indicating RW access is available.

7.3.2 error_base

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

EXPRESS specification:

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

7.3.3 time_stamp

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

EXPRESS specification:

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