TECHNICAL SPECIFICATION

ISO/TS 10303-35

First edition
2003-12-15

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

Part 35: Conformance testing methodology and framework: Abstract test methods for standard data access interface (SDAI) implementations

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

Partie 35: Méthodologie et cadre pour les essais de conformité: Méthodes d'essai abstraites pour mises en application SDAI

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Reference number
ISO/TS 10303-35:2003(E)

© ISO 2003

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003

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 either ISO at the address below or ISO's member body in the country of the requester.

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.org
Web www.iso.org

Published in Switzerland

Contents

Page

1 Scope 1
2 Normative references 1
3 Terms, definitions, and abbreviations 2
3.1 Terms defined in ISO 10303-1 2
3.2 Terms defined in ISO 10303-22 2
3.3 Terms defined in ISO 10303-31 3
3.4 Other terms and definitions 4
3.5 Abbreviations 4
4 Overview and characteristics 4
4.1 Testing characteristics 5
5 Testing process 5
5.1 Preparation for testing 6
5.2 Test campaign 6
5.3 Test conclusion 6
5.4 Test report production 6
6 SDAI abstract test schema 7
6.1 Introduction 7
6.2 SDAI abstract test schema function and procedure definitions 8
7 SDAI operation schema 156
7.1 SDAI operation schema constant definitions 157
7.2 SDAI operation schema function and procedure definitions 158
8 EXPRESS structure test schema 188

Annex A (normative) Information object registration 191

Index 192

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

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 2.

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 after three years in order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn. If the ISO/PAS or ISO/TS is confirmed, it is reviewed again after a further three years, at which time it must either be transformed into an International Standard or be withdrawn.

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

ISO/TS 10303-35 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

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

Each part of ISO 10303 is a member of one of the following series: description methods, implementation methods, conformance testing methodology and framework, integrated generic resources, integrated application resources, application protocols, abstract test suites, application interpreted constructs, and application modules. This part is a member of the conformance testing ethodology and framework series.

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

```
http://www.tc184-sc4.org/titles/
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Introduction

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

This part of ISO 10303 is a member of the conformance testing methodology and framework series. It specifies the abstract test methods for SDAI implementations. SDAI is the standard data access interface specification to data that has been defined using ISO 10303-11. SDAI is specified in ISO 10303-22. This part follows the general concepts of conformance testing defined in ISO 10303-31.

Major subdivisions in this part of ISO 10303 are:

- Abstract test cases, groups, suite and verdict criteria in clause 6;
- SDAI operations wrapped in EXPRESS procedures and functions with verdict criteria in clause 7;
- The EXPRESS structure test schema *ESTS* as the target for the abstract test cases in clause 8.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

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

Part 35: Conformance testing methodology and framework: Abstract test methods for standard data access interface (SDAI) implementations

1 Scope

This part of ISO 10303 specifies the abstract test methods and requirements for conformance testing of an implementation of a language binding of the SDAI. Since the SDAI is specified independently of any programming language, the abstract test methods presented in this part are applicable to all SDAI language bindings. The abstract test methods support as well the various implementation classes as specified in ISO 10303-22.

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

— abstract test methods for software systems that implement the SDAI;

— the specification, in a manner that is independent of any SDAI language binding, of the methods and approaches for testing of various SDAI operations;

— the specification and documentation of abstract test cases.

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

— the development of test data and/or test programs for specific language bindings;

— the specification of test methods, algorithms, or programs for the conformance testing of applications that interact with SDAI implementations;

— the architecture and implementation approach for a conformance test system that realizes the test methods specified in this part of ISO 10303.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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*

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

ISO 10303-21:2002, *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*

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

ISO 10303-32:1998, *Industrial automation systems and integration — Product data representation and exchange* — Part 32: *Conformance testing methodology and framework: Requirements on testing laboratories and clients*

3 Terms, definitions, and abbreviations

3.1 Terms defined in ISO 10303-1

For the purpose of this document, the following terms defined in ISO 10303-1 apply:

— abstract test suite;

— application protocol;

— conformance class;

— implementation method;

— PICS proforma;

— protocol implementation conformance statement (PICS).

3.2 Terms defined in ISO 10303-22

For the purpose of this document, the following terms defined in ISO 10303-22 apply:

— application schema;

— implementation class;

— repository;

— schema instance;

— SDAI language binding;

— SDAI-model;

— session;

— validation.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

3.3 Terms defined in ISO 10303-31

For the purpose of this document, the following terms defined in ISO 10303-31 apply:

— abstract test case (ATC);

— abstract test group;

— abstract test method;

— conformance;

— conformance log;

— (conformance) test report;

— conformance testing;

— executable test case;

— executable test suite;

— fail (verdict);

— implementation under test (IUT);

— inconclusive (verdict);

— pass (verdict);

— PIXIT proforma;

— Protocol Implementation eXtra Information for Testing (PIXIT);

— test campaign;

— test case error;

— testing laboratory;

— test purpose;

— test report;

— (test) verdict;

— verdict criteria.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

3.4 Other terms and definitions

For the purpose of this document, the following definitions apply:

3.4.1
abstract test operation
a function or procedure that encapsulate an SDAI operation to test its proper behaviour, including the possible error code and error base. An abstract test operation is a verdict criterion operation

3.4.2
executable test operation
an instantiation of an abstract test operation for a particular programming language and SDAI implementation.

3.4.3
SDAI operation
an operation defined in clause 10 of ISO10303-22:1998

3.4.4
verdict criterion operation
a function or procedure defining a verdict criterion

3.5 Abbreviations

For the purposes of this document, the following abbreviations apply.

IUT	Implementation Under Test
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation eXtra Information for Testing
SDAI	Standard Data Access Interface, ISO 10303-22
ESTS	EXPRESS Structure Test Schema

4 Overview and characteristics

This part of ISO 10303 describes abstract test methods that conformance test systems would implement in order to test SDAI implementations. General principles and an overall framework for conformance testing are provided in ISO 10303-31. Requirements on test laboratories are defined in ISO 10303-32. The methods for preparing, controlling, observing and analysing implementations during testing are defined in this part of ISO 10303.

Abstract test methods are given for the SDAI implementation classes 1 to 7. An abstract test method is a set of instructions on how to apply specified abstract test cases for particular SDAI implementations. The abstract test cases are grouped into abstract test groups with the SDAI abstract test suite at the top level. The abstract test cases adjust themselves to the actual implementation class by accessing the information in the object, representing an instance of entity type *implementation* (see ISO 10303-22, 7.4.2).

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

The abstract test suite for SDAI implementations is defined by EXPRESS functions and procedures. For conformance testing an executable test suite has to be derived from the abstract test suite, suitable for the underlying programming language of the SDAI implementation.

NOTE This part of ISO 10303 does not specify how to derive an executable test suite from the abstract test suite. This is the task of a testing laboratory or its client.

Clause 8 defines the EXPRESS Structure Test Schema (ESTS) as the basic application schema operated by the IUT. The abstract test suite in clause 6 specifies test cases and verdict criteria for particular entities and attributes of the ESTS. The test cases and verdict criteria refer to the "wrapper" functions and procedures for the SDAI operations in clause 6.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

4.1 Testing characteristics

The characteristics of the test methods specified in this part of ISO 10303 are:

— An SDAI implementation may be early bound, late bound, or both. Test methods specified in this part of ISO 10303 address all such implementations;

— An SDAI implementation obeys the state model described in ISO 10303-22. The specified tests ensure this;

— ISO 10303-22 is written independent of any programming language. To keep the language independence this part of ISO 10303 is using EXPRESS to specify the abstract tests;

— SDAI operations provide means for data manipulation and transactions. Testing these operations will provide assurance of their correctness and whether they had the desired effect on the persistent storage. These operations include create, delete, modify, validate and various manipulation operations that act on schema instances, SDAI-models, and instances of application schema entities;

— In situations of error, SDAI operations return error codes. Testing will encompass all reasonable error situations for an operation to ensure that appropriate error codes are returned;

— The testing of error handling requires checking the returned error value against the actual error condition that triggered that return value;

— Environment and session operations provide the capability for changing the state of the SDAI session. Conformance testing will ascertain that only the permitted transitions take place and that only permitted operations for a state are allowed;

— Aggregate operations play important roles in SDAI implementations, facilitating the successful completion of other complex operations or sequences of operations;

— Test purposes and verdicts criteria are provided for all test cases;

— Specification of the conformance log along with the test purpose and verdict criteria definitions.

5 Testing process

The testing process consists of the preparation of the test, running the campaign, making a conclusion and producing the final test report.

5.1 Preparation for testing

PICS and PIXIT proformas are completed by IUT vendors prior to testing. The PICS shall be based on the PICS proforma of ISO 10303-22, Annex B. The PICS shall further specify:

— the SDAI language binding;

— the binding style: late binding and/or early binding as defined in the particular SDAI language binding.

An executable test suite, derived from the abstract test suite in clause 6 and the abstract test operations in clause 7 shall be available.

The application schema ESTS shall be made available to the IUT.

NOTE The way how an application schema is made available to an SDAI implementation is usually implementation dependent.

The SDAI entity **implementation** contains information that shall match to the PICS. This information is printed into the conformance log by the abstract test case *atc_implementation* (see 6.3).

5.2 Test campaign

A conformance test campaign is a sequenced execution of all required executable test cases (ETCs) starting from the executable test suite. The results of this sequenced execution determines conformance.

Modifications to the IUT are not permitted during a test campaign. Modifications to the ETCs or to the sequence of their execution is not permitted during a test campaign, except in the situation where the ETC is determined to be in error.

If an ETC is determined to be in error, a verdict of INCONCLUSIVE shall be assigned to its execution until the error is resolved and the test repeated.

5.3 Test conclusion

A test campaign may terminate for any reason. A normal termination of a test campaign occurs when all its executable tests have been run. A PASS verdict shall be assigned to a campaign if all the tests of the campaign have returned PASS verdicts and no violation of any ISO 10303 part is detected.

5.4 Test report production

A conformance test report shall be created after a test campaign terminates. In addition to the requirements specified in ISO 10303-31 and ISO 10303-32 this report shall contain:

— used programming language;

— type of tested SDAI language binding, late binding or early binding;

— tested SDAI implementation classes;

— the executable test suite being used;

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

— a detailed conformance log as specified in clause 6.

Any relevant information on the testing environment shall be included as well.

6 SDAI abstract test schema

This clause specifies the SDAI_abstract_test_schema containing the abstract test cases for SDAI implementations formulated in a hierarchical manner. A test campaign starts with the *SDAI_abstract_test_suite* procedure (see 6.1). From this root node other abstract test groups and abstract test cases are invoked.

The following EXPRESS specification begins the **SDAI abstract test schema**.

EXPRESS specification:

```
*)
SCHEMA SDAI_abstract_test_schema;

USE FROM SDAI_dictionary_schema; -- ISO 10303-22

USE FROM SDAI_population_schema; -- ISO 10303-22

USE FROM SDAI_session_schema;  -- ISO 10303-22

USE FROM SDAI_parameter_data_schema;  -- ISO 10303-22

REFERENCE FROM SDAI_operation_schema; -- ISO 10303-35

USE FROM ESTS;  -- ISO 10303-35

(*
```

NOTE The schemas referenced above are specified in the following parts of ISO 10303:

SDAI_dictionary_schema	ISO 10303-22
SDAI_population_schema	ISO 10303-22
SDAI_session_schema	ISO 10303-22
SDAI_parameter_data_schema	ISO 10303-22
SDAI_ operation_schema	Clause 7 of this part of ISO 10303
ESTS	Clause 8 of this part of ISO 10303

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.1 Introduction

The subject of the SDAI_abstract_test_schema is to define **abstract test cases** for testing SDAI implementations. An abstract test case specifies one or several test purposes, verdict criterion operations and verdict statements. The structure of an abstract test case is as follows:

— preparation of test data;

— writing the name of the abstract test case in the conformance log with the *atc* procedure (see 6.164);

— writing the description of a test purpose to the conformance log with the *purpose* procedure (see 1.165);

— performing one or several verdict criterion operations. The verdict criterion operations are *assert* (see 6.2.162), *check_instance* (see 6.2.163), and the executable test operations (see 7.2);

— assigning a verdict of the test purpose, based on the results of the verdict criteria. This is accomplished by invoking the *verdict* procedure (see 6.2.166) that writes the verdict result to the conformance log.

Some abstract test cases are dependent of the supported implementation levels, as defined in the *implementation* object. Whenever an SDAI operation is dependent on a specific implementation level, the EXPRESS algorithms query this information in order to determine if the test case is applicable to the implementation under test.

The abstract test cases specified in this clause are built around SDAI operations specified in ISO 10303-22. For conformance testing executable test cases shall be derived from the abstracted test cases to build up an equivalent executable test suite. Since there exists no guaranteed one-to-one relationship between operations in ISO 10303-22 and a particular SDAI language binding, the statements of a executable test case may not be identical to those in the abstract test case. However the statements of an executable test case shall reflect the logical structure of its abstract test case, defined by the test purposes and the specified verdict criteria. The executable test cases shall also follow the specifications of the corresponding SDAI language bindings defined in other parts of ISO 10303. SDAI language bindings may specify modified or extended functionality for some SDAI operations to that defined in ISO 10303-22. In the case that the specifications in an SDAI language binding conflicts with the ones given in ISO 10303-22, the executable test cases shall be adopted for the specifications of the SDAI language bindings.

Attributes of the session and population entities sdai_session, sdai_transaction, implementation, sdai_repository, sdai_repository_contents, sdai_model, sdai_model_contents, entity_extent and schema_instance are accessed by usual EXPRESS expressions. All application entities and attributes of the ESTS schema are accessed by the abstract test operations only.

6.2 SDAI abstract test schema function and procedure definitions

The root of the **abstract test suite** is given by the *SDAI_abstract_test_suite* procedure (see 6.2.1). From this root node other procedures and functions, representing the **abstract test groups** and **abstract test cases** are invoked.

Abbreviated names are used in the identifiers of the functions and procedures declared in this schema. Prefixes used in these identifiers have the following meanings:

atg	abstract test group
atc	abstract test case

Further auxiliary functions and procedures are defined to prepare test data, generate the conformance log and establish the verdict criteria.

Functions with the prefix 'macro' are used to prepare the IUT for a specific state. The specified macros are:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

— macro_get_closed_repository (see 6.2.147);

— macro_get_open_repository (see 6.2.148);

— macro_get_schema_instance (see 6.2.149);

— macro_get_sdai_model_unset_mode (see 6.2.150);

— macro_get_sdai_model_read_only (see 6.2.151);

— macro_get_sdai_model_read_write (see 6.2.152);

— macro_get_sdai_model_read_write_different (see 6.2.153);

— macro_get_data_dictionary_model (see 6.2.154);

— macro_get_entity_extent (see 6.2.155);

— macro_check_extent_if_populated (see 6.2.156);

— macro_check_instance_if_values_unset (see 6.2.157);

— macro_compare_aggregates (see 6.2.158);

— macro_convert_primitive_to_aggregate (see 6.2.159);

— macro_clear_aggregate (see 6.2.160).

The *asp* function (see 6.2.161) is used to construct values of type assignable_primitive for testing.

Besides the abstract test operations defined in clause 7 the following procedures participate in the preparation for the verdict criteria used to establish the verdict:

— assert (see 6.2.162);

— check_instance (see 6.2.163);

— purpose (see 6.2.165);

— verdict (see 6.2.166).

The following procedures are used to generate the conformance log:

— atc (see 6.2.164);

— purpose (see 6.2.165);

— verdict (see 6.2.166);

— print (see 6.2.167).

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.1 SDAI_abstract_test_suite

The SDAI abstract test suite shall be used for the assessment of an SDAI implementation.

EXPRESS specification:

```
*)
PROCEDURE SDAI_abstract_test_suite;
  LOCAL
    session : sdai_session;
    repo : sdai_repository;
    schema_inst : schema_instance;
    modl, modl2 : sdai_model;
  END_LOCAL;

  -- tests on sdai_session
  session := atc_open_session;
  atc_implementation(session);
  repo := macro_get_closed_repository;
  atc_close_session(repo);

  modl := macro_get_sdai_model_read_write;
  atc_create_non_persistent_list(modl);
  close_session(session, NO_ERROR, ?);

  session := open_session(NO_ERROR, ?);
  if (session.sdai_implementation.transaction_level = 3) then
    atc_start_transaction_read_only_access(session);
  end_if;
  close_session(session, NO_ERROR, ?);

  session := open_session(NO_ERROR, ?);
  if (session.sdai_implementation.transaction_level = 3) then
    modl := macro_get_sdai_model_read_write;
    atc_commit(modl);
    atc_abort(modl);
  end_if;
  close_session(session, NO_ERROR, ?);

  -- tests on sdai_repository
  repo := macro_get_closed_repository;
  atc_open_repository(repo);
  close_session(session, NO_ERROR, ?);

  repo := macro_get_closed_repository;
  atc_create_sdai_model(repo);
  close_session(session, NO_ERROR, ?);

  repo := macro_get_open_repository;
  atc_create_schema_instance(repo);
  close_session(session, NO_ERROR, ?);

  repo := macro_get_open_repository;
  atc_close_repository(repo);
  close_session(session, NO_ERROR, ?);

  -- tests on schema_instance
  schema_inst := macro_get_schema_instance;
  atc_add_sdai_model(schema_inst);
  close_session(session, NO_ERROR, ?);

  schema_inst := macro_get_schema_instance;
  atc_remove_sdai_model(schema_inst);
  close_session(session, NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
schema_inst := macro_get_schema_instance;
atc_rename_schema_instance(schema_inst);
close_session(session, NO_ERROR, ?);

schema_inst := macro_get_schema_instance;
atc_delete_schema_instance(schema_inst);
close_session(session, NO_ERROR, ?);

-- tests on sdai_model
modl := macro_get_sdai_model_unset_mode;
atc_start_read_only_access(modl);
close_session(session, NO_ERROR, ?);

modl := macro_get_sdai_model_unset_mode;
atc_promote_sdai_model_to_read_write(modl);
close_session(session, NO_ERROR, ?);

modl := macro_get_sdai_model_read_only;
atc_end_read_only_access(modl);
close_session(session, NO_ERROR, ?);

modl := macro_get_sdai_model_unset_mode;
atc_start_read_write_access(modl);
close_session(session, NO_ERROR, ?);

modl := macro_get_sdai_model_read_write;
atc_end_read_write_access(modl);
close_session(session, NO_ERROR, ?);

modl := macro_get_sdai_model_read_write;
modl2 := macro_get_sdai_model_read_write_different(modl);
atc_delete_SDAI_model(modl, modl2);
close_session(session, NO_ERROR, ?);

modl := macro_get_sdai_model_read_write;
atc_rename_SDAI_model(modl);
close_session(session, NO_ERROR, ?);

modl := macro_get_sdai_model_read_write;
atc_get_entity_definition(modl);
close_session(session, NO_ERROR, ?);

repo := macro_get_open_repository;
atc_create_entity_instance(repo);
close_session(session, NO_ERROR, ?);

if (session.sdai_implementation.transaction_level = 2) then
   repo := macro_get_open_repository;
   atc_undo_changes(repo);
   atc_save_changes(repo);
   close_session(session, NO_ERROR, ?);
end_if;

-- tests on entity type
session := open_session(NO_ERROR, ?);
if (session.sdai_implementation.expression_level >= 2) then
   atc_get_complex_entity_definition;
end_if;
atc_is_subtype_of;
close_session(session, NO_ERROR, ?);

-- tests on application entity instance
modl := macro_get_sdai_model_read_write;
atg_attribute_basic(modl);
close_session(session, NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   modl := macro_get_sdai_model_read_write;
   modl2 := macro_get_sdai_model_read_write_different(modl);
   atc_attribute_entity_instance_in_another_model(modl, modl2);
   close_session(session, NO_ERROR, ?);

   modl := macro_get_sdai_model_read_write;
   modl2 := macro_get_sdai_model_read_write_different(modl);
   atc_attribute_entity_instance_in_closed_repository(modl, modl2);
   close_session(session, NO_ERROR, ?);

   modl := macro_get_sdai_model_read_write;
   atc_find_entity_instance_sdai_model(modl);
   atc_get_instance_type(modl);
   atc_is_instance_of(modl);
   atc_is_kind_of(modl);
   atc_persistent_label_and_session_identifier(modl);
   close_session(session, NO_ERROR, ?);

   modl := macro_get_sdai_model_read_write;
   atc_find_entity_instance_users(modl);
   atc_find_entity_instance_usedin(modl);
   atc_find_instance_roles(modl);
   close_session(session, NO_ERROR, ?);

   modl := macro_get_sdai_model_read_write;
   atc_copy_application_instance(modl);
   close_session(session, NO_ERROR, ?);

   modl := macro_get_sdai_model_read_write;
   atc_delete_application_instance(modl);
   close_session(session, NO_ERROR, ?);

   if (session.sdai_implementation.expression_level >= 2) then
      modl := macro_get_sdai_model_read_write;
      atc_validate_required_explicit_attributes_assigned(modl);
      atc_validate_inverse_attributes(modl);
      atc_validate_explicit_attributes_references(modl);
      atc_validate_aggregates_size(modl);
      atc_validate_aggregates_uniqueness(modl);
      atc_validate_array_not_optional(modl);
      close_session(session, NO_ERROR, ?);
   end_if;

   -- tests on aggregate
   modl := macro_get_sdai_model_read_write;
   atg_aggregate_simple(modl);
   close_session(session, NO_ERROR, ?);

   modl := macro_get_sdai_model_read_write;
   atg_nested_aggregate(modl);
   close_session(session, NO_ERROR, ?);

END_PROCEDURE; -- SDAI_abstract_test_suite
(*
```

6.2.2 atc_open_session

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *open session*. The opened **sdai_session** is returned.

EXPRESS specification:

```
*)
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
FUNCTION atc_open_session : sdai_session;
   LOCAL
      session : sdai_session;
   END_LOCAL;

   atc('atc_open_session');

   purpose('Ensures that open_session works correctly when '
      + 'SDAI session is closed');
   session := open_session(NO_ERROR, ?);
   verdict;

   return (session);

END_FUNCTION; -- atc_open_session
(*
```

6.2.3 atc_implementation

This abstract test case shall be used for the assessment of the implementation of the SDAI entity *implementation*. The **implementation** information is printed to the conformance log. The printed *implementation class*, *transaction level*, *expression level* and *domain equivalence level* shall be identical to the PICS. The *recording level* and *scope level* is not checked by this part of ISO 10303.

EXPRESS specification:

```
*)
PROCEDURE atc_implementation(session : sdai_session);
   LOCAL
      imp : implementation := session.sdai_implementation;
   END_LOCAL;

   print('Implementation Under Test (IUT):');
   print(' name = ' + imp.name);
   print(' level = ' + imp.level);
   print(' sdai version = ' + imp.sdai_version);
   print(' binding version = ' + imp.binding_version);
   print(' implementation class = ' + format(imp.implementation_class,''));
   print(' transaction level = ' + format(imp.transaction_level,''));
   print(' expression level = ' + format(imp.expression_level,''));
   print(' recording level = ' + format(imp.recording_level,''));
   print(' scope level = ' + format(imp.scope_level,''));
   print(' domain equivalence level = ' +
           format(imp.domain_equivalence_level,''));

   atc('atc_implementation');

   purpose('Ensure the correct sdai_version');
   assert(imp.sdai_version = '{ iso standard 10303 part(22) version(0) }');
   verdict;

   purpose('Ensure that the implementation class is valid');
   assert(imp.implementation_class >= 1);
   assert(imp.implementation_class <= 7);
   verdict;

   CASE imp.implementation_class OF
   1 : BEGIN
         purpose('Ensure proper transaction level');
         assert(imp.transaction_level = 1);
         verdict;

         purpose('Ensure proper expression level');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
         assert(imp.expression_level >= 1);
         verdict;

         purpose('Ensure proper domain equivalence level');
         assert(imp.domain_equivalence_level >= 1);
         verdict;
      END;
   2 : BEGIN
         purpose('Ensure proper transaction level');
         assert(imp.transaction_level = 1);
         verdict;

         purpose('Ensure proper expression level');
         assert(imp.expression_level >= 2);
         verdict;

         purpose('Ensure proper domain equivalence level');
         assert(imp.domain_equivalence_level >= 1);
         verdict;
      END;
   3 : BEGIN
         purpose('Ensure proper transaction level');
         assert(imp.transaction_level = 1);
         verdict;

         purpose('Ensure proper expression level');
         assert(imp.expression_level >= 3);
         verdict;

         purpose('Ensure proper domain equivalence level');
         assert(imp.domain_equivalence_level >= 2);
         verdict;
      END;
   4 : BEGIN
         purpose('Ensure proper transaction level');
         assert(imp.transaction_level = 2);
         verdict;

         purpose('Ensure proper expression level');
         assert(imp.expression_level >= 2);
         verdict;

         purpose('Ensure proper domain equivalence level');
         assert(imp.domain_equivalence_level >= 1);
         verdict;
      END;
   5 : BEGIN
         purpose('Ensure proper transaction level');
         assert(imp.transaction_level = 3);
         verdict;

         purpose('Ensure proper expression level');
         assert(imp.expression_level >= 2);
         verdict;

         purpose('Ensure proper domain equivalence level');
         assert(imp.domain_equivalence_level >= 1);
         verdict;
      END;
   6 : BEGIN
         purpose('Ensure proper transaction level');
         assert(imp.transaction_level = 3);
         verdict;

         purpose('Ensure proper expression level');
         assert(imp.expression_level >= 3);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
         verdict;

         purpose('Ensure proper domain equivalence level');
         assert(imp.domain_equivalence_level >= 2);
         verdict;
      END;
   7 : BEGIN
         purpose('Ensure proper transaction level');
         assert(imp.transaction_level = 3);
         verdict;

         purpose('Ensure proper expression level');
         assert(imp.expression_level >= 4);
         verdict;

         purpose('Ensure proper domain equivalence level');
         assert(imp.domain_equivalence_level >= 2);
         verdict;
      END;
   END_CASE;

END_PROCEDURE; -- atc_implementation
(*
```

Argument definitions:

session: (input) An open SDAI session.

6.2.4 atc_close_session

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *close session*.

EXPRESS specification:

```
*)
PROCEDURE atc_close_session(repo : sdai_repository);
   LOCAL
      session : sdai_session := repo.session;
   END_LOCAL;

   atc('atc_close_session');

   purpose('Ensures that close_session works correctly when session '
      + 'is open');
   close_session(session, NO_ERROR, ?);
   verdict;

   purpose('ensures that repositories are not available after '
      + 'session was closed');
   open_repository(session, repo, RP_NEXS, ?);
   verdict;

   purpose('ensures that close_session reports an SS_NOPN error when '
      + 'SDAI session is closed');
   close_session(session, SS_NOPN, ?);
   verdict;

END_PROCEDURE; -- atc_close_session
(*
```

Argument definitions:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

repo: (input) A closed repository (that is, from the known_servers but not active_servers set of the SDAI session).

6.2.5 atc_create_non_persistent_list

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *create non-persistent list.*

EXPRESS specification:

```
*)
PROCEDURE atc_create_non_persistent_list(modl : sdai_model);
  LOCAL
    session : sdai_session := modl.repository.session;
    non_persist_list : non_persistent_list_instance;
    aggr_created : aggregate_instance;
    iter : iterator;
    inst : application_instance :=
      create_entity_instance('ESTS.EPSILON', modl, NO_ERROR, ?);
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_create_non_persistent_list');

  purpose('Ensures that create_non_persistent_list works correctly');
  non_persist_list := create_non_persistent_list(NO_ERROR, ?);
  verdict;

  purpose('Ensures that a non-persistent list is empty after '
    + 'its creation');
  assert(get_member_count(non_persist_list, NO_ERROR, ?) = 0);
  verdict;

  purpose('Prevents assigning a non-persistent list to an attribute of '
    + 'some entity');
  put_attribute(inst, 'ESTS.EPSILON.E2', non_persist_list, VT_NVLD, ?);
  verdict;

  purpose('Ensures that an AI_NVLD error is reported when SET '
    + 'and BAG operations are applied to a non-persistent list');
  add_unordered(non_persist_list, asp(inst, ?),
    AI_NVLD, non_persist_list);
  remove_unordered(non_persist_list, asp(inst, ?),
    AI_NVLD, non_persist_list);
  aggr_created := create_aggregate_instance_unordered(non_persist_list, ?,
    AI_NVLD, non_persist_list);
  verdict;

  purpose('Ensures that an AI_NVLD error is reported when ARRAY '
    + 'operations are applied to a non-persistent list.');
  bool := test_by_index(non_persist_list, 1, AI_NVLD, non_persist_list);
  unset_value_by_index(non_persist_list, 1, AI_NVLD, non_persist_list);
  iter := create_iterator(non_persist_list, NO_ERROR, ?);
  bool := test_current_member(iter, AI_NVLD, non_persist_list);
  unset_value_current_member(iter, AI_NVLD, non_persist_list);
  verdict;

  purpose('Ensures that an AI_NVLD error is reported when any '
    + 'aggregate creation operations disallowed for non-persistent '
    + 'lists are applied.');
  aggr_created := create_aggregate_instance_by_index(non_persist_list,
    1, ?, AI_NVLD, non_persist_list);
  aggr_created := create_aggregate_instance_as_current_member(iter, ?,
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
      AI_NVLD, non_persist_list);
   aggr_created := create_aggregate_instance_before_current_member(iter, ?,
      AI_NVLD, non_persist_list);
   aggr_created := create_aggregate_instance_after_current_member(iter, ?,
      AI_NVLD, non_persist_list);
   aggr_created := add_aggregate_instance_by_index(non_persist_list, 1, ?,
      AI_NVLD, non_persist_list);
   verdict;

   purpose('Ensures that non-persistent list does not exist after '
      + 'SDAI session is closed.');
   close_session(session, NO_ERROR, ?);
   add_by_index(non_persist_list, 1, asp(inst, ?), SS_NOPN, ?);
   verdict;

END_PROCEDURE; -- atc_create_non_persistent_list
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.6 atc_start_transaction_read_only_access

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *start transaction read-only access.*

EXPRESS specification:

```
*)
PROCEDURE atc_start_transaction_read_only_access(session : sdai_session);
   LOCAL
      transaction : sdai_transaction;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_start_transaction_read_only_access');

   purpose('Ensures that start_transaction_read_only_access works '
      + 'correctly when session is open and transaction does not exist.');
   transaction := start_transaction_read_only_access(session, NO_ERROR, ?);
   verdict;

   purpose('Ensures that started transaction belongs to '
      + 'the active_transaction set of the session.');
   bool := transaction IN session.active_transaction;
   assert(bool);
   verdict;

   purpose('Ensures that start_transaction_read_only_access reports '
      + 'a TR_EXS error when transaction is already started.');
   transaction := start_transaction_read_only_access(session, TR_EXS,
        transaction);
   verdict;

END_PROCEDURE; -- atc_start_transaction_read_only_access
(*
```

Argument definitions:

session: (input) An open SDAI session.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.7 atc_commit

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *commit.*

EXPRESS specification:

```
*)
PROCEDURE atc_commit(modl : sdai_model);
  LOCAL
    repo : sdai_repository := modl.repository;
    session : sdai_session := repo.session;
    transaction : sdai_transaction := session.active_transaction[1];
    inst1, inst2 : application_instance;
    date : STRING;
    label : STRING;
    mode : access_type;
  END_LOCAL;

  atc('atc_commit');

  (* Modifying the SDAI-model under consideration by creating some
     entity instance *)
  inst1 := create_entity_instance('ESTS.OMEGA', modl, NO_ERROR, ?);

  (* Saving some data needed for future checkings. *)
  date := modl.change_date;
  label := get_persistent_label(inst1, NO_ERROR, ?);
  mode := transaction.mode;

  purpose('Ensures that commit works correctly.');
  commit(transaction, NO_ERROR, ?);
  verdict;

  purpose('Ensures that mode for transaction continues to be set.');
  assert(mode = transaction.mode);
  verdict;

  purpose('Ensures that the attribute change_date in the '
    + 'SDAI-model modified is set to the new date.');
  assert(date <> modl.change_date);
  verdict;

  purpose('Ensures that commit made the changes done to the contents '
    + 'of the open repository persistent.');
  close_repository(repo, NO_ERROR, ?);
  open_repository(session, repo, NO_ERROR, ?);
  inst2 := get_session_identifier(label, repo, NO_ERROR, ?);
  assert(inst2 :=: inst1);
  verdict;

END_PROCEDURE; -- atc_commit
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.8 atc_abort

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *abort.*

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_abort(modl : sdai_model);
  LOCAL
    transaction : sdai_transaction :=
                  modl.repository.session.active_transaction[1];
    inst : application_instance;
    mode : access_type;
    p : primitive;
  END_LOCAL;

  atc('atc_abort');

  (* Modifying the SDAI-model under consideration by creating some entity
     instance and setting one of its attributes. *)
  inst := create_entity_instance('ESTS.OMEGA', modl, NO_ERROR, ?);
  put_attribute(inst, 'ESTS.OMEGA.O1', asp(3.4, ?), NO_ERROR, ?);
  commit(transaction, NO_ERROR, ?);
  put_attribute(inst, 'ESTS.OMEGA.O1', asp(15.5, ?), NO_ERROR, ?);

  (* Saving transaction mode needed for future checking. *)
  mode := transaction.mode;

  purpose('Ensures that abort works correctly.');
  abort(transaction, NO_ERROR, ?);
  verdict;

  purpose('Ensures that mode for transaction continues to be set.');
  assert(mode = transaction.mode);
  verdict;

  purpose('Ensures that the old contents of the repository was '
    + 'restored.');
  p := get_attribute(inst, 'ESTS.OMEGA.O1', NO_ERROR, ?);
  assert(p = asp(3.4, ?));
  verdict;

END_PROCEDURE; -- atc_abort
(*
```

<u>Argument definitions:</u>

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.9 atc_open_repository

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *open repository*.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_open_repository(repo : sdai_repository);
  LOCAL
    session : sdai_session := repo.session;
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_open_repository');

  purpose('Ensures that open_repository works correctly when '
    + 'parameters are correct.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   open_repository(session, repo, NO_ERROR, ?);
   verdict;

   purpose('Ensures that opened repository belongs to the '
      + 'active_servers set of the session.');
   bool := repo IN session.active_servers;
   assert(bool);
   verdict;

   purpose('Ensures that open_repository reports an RP_OPN error '
      + 'when repository is already open.');
   open_repository(session, repo, RP_OPN, repo);
   verdict;

END_PROCEDURE; -- atc_open_repository
(*
```

<u>Argument definitions:</u>

repo: (input) A closed repository (that is, from the known_servers but not active_servers set of the SDAI session).

6.2.10 atc_create_sdai_model

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *create SDAI-model*.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_create_sdai_model(repo : sdai_repository);
   LOCAL
      session : sdai_session := repo.session;
      transaction : sdai_transaction;
      modl : sdai_model;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_create_sdai_model');

   if (session.sdai_implementation.transaction_level = 3) then
      (* Starting transaction read-only access. *)
      transaction := start_transaction_read_only_access(session,
         NO_ERROR, ?);

      purpose('Ensures that create_sdai_model reports a TR_NRW error '
         + 'when transaction is not read-write.');
      modl := create_sdai_model(repo, 'exceptional_name_within_repo',
         'ESTS', TR_NRW, transaction);
      verdict;

      purpose('Starting transaction read-write access.');
      end_transaction_access_and_abort(transaction, NO_ERROR, ?);
      transaction := start_transaction_read_write_access(session,
         NO_ERROR, ?);
      verdict;

   end_if;

   purpose('Ensures that create_sdai_model reports an RP_NOPN error '
      + 'when repository  within which an SDAI-model is to be '
      + 'created is closed.');
   modl := create_sdai_model(repo, 'exceptional_name_within_repo', 'ESTS',
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      RP_NOPN, repo);
   verdict;

   (* Opening the repository. *)
   open_repository(session, repo, NO_ERROR, ?);

   purpose('Ensures that create_sdai_model reports a VA_NSET error '
      + 'when the name of an SDAI-model is not submitted.');
   modl := create_sdai_model(repo, ?, 'ESTS', VA_NSET, ?);
   verdict;

   purpose('Ensures that create_sdai_model reports an SD_NDEF error '
      + 'when schema definition is not submitted.');
   modl := create_sdai_model(repo, 'exceptional_name_within_repo', ?,
      SD_NDEF, ?);
   verdict;

   purpose('Ensures that create_sdai_model works correctly '
      + 'when all parameters are correct.');
   modl := create_sdai_model(repo, 'exceptional_name_within_repo', 'ESTS',
      NO_ERROR, ?);
   verdict;

   purpose('Ensures that created SDAI-model has no access mode.');
   end_read_only_access(modl, MX_NDEF, modl);
   verdict;

   purpose('Ensures that created SDAI-model belongs to the models '
      + 'set of the sdai_repository_contents.');
   bool := modl IN repo.contents.models;
   assert(bool);
   verdict;

   purpose('Ensures that create_sdai_model reports a MO_DUP error '
      + 'when an SDAI-model with the provided name already exists.');
   modl := create_sdai_model(repo, 'exceptional_name_within_repo', 'ESTS',
      MO_DUP, modl);
   verdict;

END_PROCEDURE; -- atc_create_sdai_model
(*
```

<ins>Argument definitions:</ins>

repo: (input) A closed repository (that is, from the known_servers but not active_servers set of the SDAI session).

6.2.11 atc_create_schema_instance

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *create schema instance.*

<ins>EXPRESS specification:</ins>

```
*)
PROCEDURE atc_create_schema_instance(repo : sdai_repository);
   LOCAL
      schema_inst : schema_instance;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_create_schema_instance');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that create_schema_instance reports a VA_NSET '
      + 'error when the name of the schema instance to be created '
      + 'is not submitted.');
   schema_inst := create_schema_instance(repo, ?, 'ESTS', VA_NSET, ?);
   verdict;

   purpose('Ensures that create_schema_instance reports an SD_NDEF '
      + 'error when schema definition is not submitted.');
   schema_inst := create_schema_instance(repo,
      'exceptional_name_within_repo', ?, SD_NDEF, ?);
   verdict;

   purpose('Ensures that create_schema_instance works correctly '
      + 'when parameters are correct.');
   schema_inst := create_schema_instance(repo,
      'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
   verdict;

   purpose('Ensures that created schema instance belongs to the schemas '
      + 'set of the sdai_repository_contents.');
   bool := schema_inst IN repo.contents.schemas;
   assert(bool);
   verdict;

   purpose('Ensures that create_schema_instance reports an SI_DUP '
      + 'error when schema instance with the provided name already '
      + 'exists.');
   schema_inst := create_schema_instance(repo,
      'exceptional_name_within_repo', 'ESTS', SI_DUP, schema_inst);
   verdict;

END_PROCEDURE; -- atc_create_schema_instance
(*
```

<u>Argument definitions:</u>

repo: (input) An open repository (that is, from the active_servers set of the SDAI session).

6.2.12 atc_close_repository

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *close repository*.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_close_repository(repo : sdai_repository);
   LOCAL
      transaction : sdai_transaction;
      modl : sdai_model := create_sdai_model(repo,
             'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
      inst : application_instance;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_close_repository');

   if (repo.session.sdai_implementation.transaction_level = 3) then

      purpose('Ensures that close_repository reports a TR_RW error when '
         + 'some SDAI-model within the repository has been created.');
      transaction := repo.session.active_transaction[1];
      close_repository(repo, TR_RW, transaction);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
      verdict;

      (* Making persistent all changes in open repositories. *)
      start_read_write_access(modl, NO_ERROR, ?);
      commit(transaction, NO_ERROR, ?);
   end_if;

   purpose('Ensures that close_repository works correctly when all '
      + 'changes are resolved.');
   close_repository(repo, NO_ERROR, ?);
   verdict;

   purpose('Ensures that closed repository does not belong to the '
      + 'active_servers set.');
   bool := repo IN repo.session.active_servers;
   assert(NOT bool);
   verdict;

   purpose('Ensures that an SDAI-model in a closed repository does '
      + 'not belong to the active_models set.');
   bool := modl IN repo.session.active_models;
   assert(NOT bool);
   verdict;

   purpose('Ensures that SDAI-models in a closed repository are no '
      + 'longer accessible.');
   inst := create_entity_instance('ESTS.ALPHA', modl, RP_NOPN, repo);
   verdict;

   purpose('Ensures that close_repository reports an RP_NOPN error '
      + 'when repository is closed.');
   close_repository(repo, RP_NOPN, repo);
   verdict;

END_PROCEDURE; -- atc_close_repository
(*
```

Argument definitions:

repo: (input) An open repository (that is, from the active_servers set of the SDAI session).

6.2.13 atc_add_sdai_model

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add SDAI-model*.

EXPRESS specification:

```
*)
PROCEDURE atc_add_sdai_model(schema_inst : schema_instance);
   LOCAL
      modl : sdai_model := create_sdai_model(schema_inst.repository,
              'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_add_sdai_model');

   purpose('Ensures that add_sdai_model reports a VA_NSET error when '
      + 'SDAI-model to be added is not submitted.');
   add_sdai_model(schema_inst, ?, VA_NSET, ?);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that add_sdai_model works correctly when '
     + 'parameters are correct.');
   add_sdai_model(schema_inst, modl, NO_ERROR, ?);
   verdict;

   purpose('Ensures that added SDAI-model belongs to associated_models '
     + 'set of the schema instance.');
   bool := modl IN schema_inst.associated_models;
   assert(bool);
   verdict;

   purpose('Ensures that schema instance belongs to associated_with set '
     + 'of the SDAI-model added.');
   bool := schema_inst IN modl.associated_with;
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_add_sdai_model
(*
```

Argument definitions:

schema_inst: (input) A schema instance whose native schema is the ESTS schema.

6.2.14 atc_remove_sdai_model

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove SDAI-model*.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_sdai_model(schema_inst : schema_instance);
  LOCAL
    modl : sdai_model := create_sdai_model(schema_inst.repository,
           'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_remove_sdai_model');

  (* Initially, some SDAI-model shall be associated with a given schema
     instance. *)
  add_sdai_model(schema_inst, modl, NO_ERROR, ?);

  purpose('Ensures that remove_sdai_model reports a VA_NSET error when '
    + 'SDAI-model to be removed is not submitted.');
  remove_sdai_model(schema_inst, ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that remove_sdai_model works correctly when '
    + 'parameters are correct.');
  remove_sdai_model(schema_inst, modl, NO_ERROR, ?);
  verdict;

  purpose('Ensures that the removed SDAI-model does not belong to '
    + 'associated_models set of the schema instance.');
  bool := modl IN schema_inst.associated_models;
  assert(NOT bool);
  verdict;

  purpose('Ensures that schema instance does not belong to '
    + 'associated_with set of the SDAI-model removed.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   bool := schema_inst IN modl.associated_with;
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_remove_sdai_model
(*
```

Argument definitions:

schema_inst: (input) A schema instance whose native schema is the ESTS schema.

6.2.15 atc_rename_schema_instance

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *rename schema instance*.

EXPRESS specification:

```
*)
PROCEDURE atc_rename_schema_instance(schema_inst : schema_instance);
   LOCAL
      schema_inst_created : schema_instance;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_rename_schema_instance');

   purpose('Ensures that rename_schema_instance reports a VA_NSET '
      + 'error when a new name is not submitted.');
   rename_schema_instance(schema_inst, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that rename_schema_instance reports a SI_DUP error '
      + 'when schema instance with a submitted name already exists '
      + 'in the repository.');
   schema_inst_created := create_schema_instance(schema_inst.repository,
      'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
   rename_schema_instance(schema_inst, 'exceptional_name_within_repo',
      SI_DUP, schema_inst_created);
   verdict;

   purpose('Ensures that rename_schema_instance works correctly when '
      + 'the name provided is different from the names of other '
      + 'schema instances in the same repository.');
   rename_schema_instance(schema_inst,
      'another_exceptional_name_within_repo', NO_ERROR, ?);
   verdict;

   purpose('Ensures that schema instance renamed stays in schemas set '
      + 'of the sdai_repository_contents.');
   bool := schema_inst IN schema_inst.repository.contents.schemas;
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_rename_schema_instance
(*
```

Argument definitions:

schema_inst: (input) A schema instance whose native schema is the ESTS schema.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.16 atc_delete_schema_instance

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *delete schema instance*.

EXPRESS specification:

```
*)
PROCEDURE atc_delete_schema_instance(schema_inst : schema_instance);
  LOCAL
    model1 : sdai_model := create_sdai_model(schema_inst.repository,
           'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
    model2 : sdai_model := create_sdai_model(schema_inst.repository,
           'another_exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_delete_schema_instance');

  (* Initially, two created SDAI-models are added to a given schema
     instance. *)
  add_sdai_model(schema_inst, model1, NO_ERROR, ?);
  add_sdai_model(schema_inst, model2, NO_ERROR, ?);

  purpose('Ensures that delete_schema_instance works correctly.');
  delete_schema_instance(schema_inst, NO_ERROR, ?);
  verdict;

  purpose('Ensures that schema instance deleted does not belong '
    + 'to the schemas set of the sdai_repository_contents.');
  bool := schema_inst IN schema_inst.repository.contents.schemas;
  assert(NOT bool);
  verdict;

  purpose('Ensures that schema instance, that was deleted, does not '
    + 'belong to associated_with set of any SDAI-model.');
  bool := schema_inst IN model1.associated_with;
  assert(bool);
  verdict;

END_PROCEDURE; -- atc_delete_schema_instance
(*
```

Argument definitions:

schema_inst: (input) A schema instance whose native schema is the ESTS schema.

6.2.17 atc_start_read_only_access

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *start read-only access*.

EXPRESS specification:

```
*)
PROCEDURE atc_start_read_only_access(modl : sdai_model);
  LOCAL
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_start_read_only_access');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that an SDAI-model with no access does not belong '
     + 'to active_models set of the SDAI session.');
   bool := modl IN modl.repository.session.active_models;
   assert(NOT bool);
   verdict;

   purpose('Ensures that start_read_only_access works correctly when '
     + 'SDAI-model access is not started.');
   start_read_only_access(modl, NO_ERROR, ?);

   purpose('Ensures that model mode is changed to read_only.');
   assert(modl.mode = read_only);
   verdict;

   purpose('Ensures that an SDAI-model with access started belongs '
     + 'to active_models set of the SDAI session.');
   bool := modl IN modl.repository.session.active_models;
   assert(bool);
   verdict;

   purpose('Ensures that start_read_only_access reports an MX_RO error '
     + 'when SDAI-model is already in read-only mode. Also this means '
     + 'that previous invocation of start_read_only_access correctly '
     + 'started the access of the SDAI-model.');
   start_read_only_access(modl, MX_RO, modl);
   verdict;

   purpose('Ensures that start_read_only_access reports an MX_RW error '
     + 'when an SDAI-model is in read-write mode.');
   promote_sdai_model_to_read_write(modl, NO_ERROR, ?);
   start_read_only_access(modl, MX_RW, modl);
   verdict;

END_PROCEDURE; -- atc_start_read_only_access
(*
```

Argument definitions:

modl: (input) An SDAI-model with unset mode, based on the ESTS schema.

6.2.18 atc_promote_sdai_model_to_read_write

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *promote SDAI-model to read-write.*

EXPRESS specification:

```
*)
PROCEDURE atc_promote_sdai_model_to_read_write(modl : sdai_model);
  LOCAL
    model_dict : sdai_model;
  END_LOCAL;

  atc('atc_promote_sdai_model_to_read_write');

  purpose('Ensures that promote_sdai_model_to_read_write reports '
    + 'an MX_NDEF error when SDAI-model access is not started.');
  promote_sdai_model_to_read_write(modl, MX_NDEF, modl);
  verdict;

  -- Preparing SDAI-model for promote_sdai_model_to_read_write operation
  start_read_only_access(modl, NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
  purpose('Ensures that promote_sdai_model_to_read_write works '
    + 'correctly when SDAI-model is in read-only mode.');
  promote_sdai_model_to_read_write(modl, NO_ERROR, ?);
  verdict;

  purpose('Ensures that promote_sdai_model_to_read_write reports an '
    + 'MX_RW error when an SDAI-model is in read-write mode. '
    + 'Also this means that previous invocation of '
    + 'promote_sdai_model_to_read_write correctly '
    + 'established the access of the SDAI-model.');
  promote_sdai_model_to_read_write(modl, MX_RW, modl);
  verdict;

  purpose('Ensures that promote_sdai_model_to_read_write reports '
    + 'an FN_NAVL error when a data dictionary SDAI-model for '
    + 'promoting is submitted');
  model_dict := macro_get_data_dictionary_model;
  promote_sdai_model_to_read_write(model_dict, FN_NAVL, ?);
  verdict;

END_PROCEDURE; -- atc_promote_sdai_model_to_read_write
(*
```

<u>Argument definitions:</u>

modl: (input) An SDAI-model with unset mode, based on the ESTS schema.

6.2.19 atc_end_read_only_access

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *end read-only access.*

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_end_read_only_access(modl : sdai_model);
  LOCAL
    model_dict : sdai_model;
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_end_read_only_access');

  purpose('Ensures that end_read_only_access works correctly when '
    + 'SDAI-model is in read-only mode.');
  end_read_only_access(modl, NO_ERROR, ?);
  verdict;

  purpose('Ensures that an SDAI-model with access ended does not belong '
    + 'to active_models set of the SDAI session.');
  bool := modl IN modl.repository.session.active_models;
  assert(NOT bool);
  verdict;

  purpose('Ensures that end_read_only_access reports an MX_NDEF error '
    + 'when SDAI-model access is not started. Also this means '
    + 'that previous invocation of end_read_only_access correctly '
    + 'ended access of the SDAI-model.');
  end_read_only_access(modl, MX_NDEF, modl);
  verdict;

  purpose('Ensures that end_read_only_access reports an MX_RW error '
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      + 'when an SDAI-model is in read-write mode.');
   start_read_write_access(modl, NO_ERROR, ?);
   end_read_only_access(modl, MX_RW, modl);
   verdict;

   purpose('Ensures that end_read_only_access reports an FN_NAVL error '
      + 'when a data dictionary SDAI-model for ending its access '
      + 'is submitted.');
   model_dict := macro_get_data_dictionary_model;
   end_read_only_access(model_dict, FN_NAVL, ?);
   verdict;

END_PROCEDURE; -- atc_end_read_only_access
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-only mode, based on the ESTS schema.

6.2.20 atc_start_read_write_access

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *start read-write access*.

EXPRESS specification:

```
*)
PROCEDURE atc_start_read_write_access(modl : sdai_model);
   LOCAL
      model_dict : sdai_model;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_start_read_write_access');

   purpose('Ensures that start_read_write_access works correctly when '
      + 'SDAI-model access is not started.');
   start_read_write_access(modl, NO_ERROR, ?);
   verdict;

   purpose('Ensures that an SDAI-model with access started belongs '
      + 'to active_models set of the SDAI session.');
   bool := modl IN modl.repository.session.active_models;
   assert(bool);
   verdict;

   purpose('Ensures that model mode is changed to read_write.');
   assert(modl.mode = read_write);
   verdict;

   purpose('Ensures that start_read_write_access reports an MX_RW '
      + 'error when SDAI-model is already in read-write mode. '
      + 'Also this means that previous invocation of '
      + 'start_read_write_access correctly started the '
      + 'access of the SDAI-model.');
   start_read_write_access(modl, MX_RW, modl);
   verdict;

   purpose('Ensures that start_read_write_access reports an MX_RO '
      + 'error when an SDAI-model is in read-only mode.');
   end_read_write_access(modl, NO_ERROR, ?);
   start_read_only_access(modl, NO_ERROR, ?);
   start_read_write_access(modl, MX_RO, modl);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   verdict;

   purpose('Ensures that start_read_write_access reports an FN_NAVL error '
      + 'when a data dictionary SDAI-model for its starting in read-write '
      + 'mode is submitted.');
   model_dict := macro_get_data_dictionary_model;
   start_read_write_access(model_dict, FN_NAVL, ?);
   verdict;

END_PROCEDURE; -- atc_start_read_write_access
(*
```

Argument definitions:

modl: (input) An SDAI-model with unset mode, based on the ESTS schema.

6.2.21 atc_end_read_write_access

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *end read-write access.*

EXPRESS specification:

```
*)
PROCEDURE atc_end_read_write_access(modl : sdai_model);
   LOCAL
      transaction : sdai_transaction;
      inst1, inst2 : application_instance;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_end_read_write_access');

   if (modl.repository.session.sdai_implementation.transaction_level
        = 3) then

      purpose('Ensures that end_read_write_access reports a TR_RW error '
         + 'when some application instance within the SDAI-model has '
         + 'been created.');
      inst1 := create_entity_instance('ESTS.OMEGA', modl, NO_ERROR, ?);
      transaction := modl.repository.session.active_transaction[1];
      end_read_write_access(modl, TR_RW, transaction);
      verdict;

      purpose('Ensures that end_read_write_access reports a TR_RW error '
         + 'when some application instance within the SDAI-model has '
         + 'been modified.');
      inst2 := create_entity_instance('ESTS.IOTA', modl, NO_ERROR, ?);
      commit(transaction, NO_ERROR, ?);
      put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst2, ?), NO_ERROR, ?);
      end_read_write_access(modl, TR_RW, transaction);
      verdict;

      purpose('Ensures that end_read_write_access reports a TR_RW error '
         + 'when some application instance within the SDAI-model has '
         + 'been deleted.');
      commit(transaction, NO_ERROR, ?);
      delete_application_instance(inst1, NO_ERROR, ?);
      end_read_write_access(modl, TR_RW, transaction);
      verdict;

      (* Commit operation is executed to make access ending available. *)
      commit(transaction, NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
  end_if;

  purpose('Ensures that end_read_write_access works correctly when '
    + 'all changes within the SDAI-model are resolved.');
  end_read_write_access(modl, NO_ERROR, ?);
  verdict;

  purpose('Ensures that an SDAI-model with access ended does not belong '
    + 'to active_models set of the SDAI session.');
  bool := modl IN modl.repository.session.active_models;
  assert(NOT bool);
  verdict;

  purpose('Ensures that end_read_write_access reports an MX_NDEF '
    + 'error when SDAI-model access is not started. Also this means '
    + 'that previous invocation of end_read_write_access correctly '
    + 'ended access of the SDAI-model.');
  end_read_write_access(modl, MX_NDEF, modl);
  verdict;

  purpose('Ensures that end_read_write_access reports an MX_RO error '
    + 'when the SDAI-model is in read-only mode.');
  start_read_only_access(modl, NO_ERROR, ?);
  end_read_write_access(modl, MX_RO, modl);
  verdict;

END_PROCEDURE; -- atc_end_read_write_access
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.22 atc_delete_SDAI_model

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *delete SDAI-model*.

EXPRESS specification:

```
*)
PROCEDURE atc_delete_SDAI_model(model1 : sdai_model; model2 : sdai_model);
  LOCAL
    repo : sdai_repository := model2.repository;
    schema_inst : schema_instance := model2.associated_with[1];
    model_dict : sdai_model;
    inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
      model1, NO_ERROR, ?);
    inst2 : application_instance := create_entity_instance('ESTS.IOTA',
      model2, NO_ERROR, ?);
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_delete_SDAI_model');

  (* Prepares an instance in one SDAI-model to reference an instance in
     another SDAI-model that will be deleted.*)
  put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst2, ?), NO_ERROR, ?);

  purpose('Ensures that delete_sdai_model works correctly.');
  delete_sdai_model(model2, NO_ERROR, ?);
  verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that deleted SDAI-model does not belong to models '
      + 'set of the sdai_repository_contents.');
   bool := model2 IN repo.contents.models;
   assert(NOT bool);
   verdict;

   purpose('Ensures that deleted SDAI-model does not belong '
      + 'to active_models set of the SDAI session.');
   bool := model2 IN repo.session.active_models;
   assert(NOT bool);
   verdict;

   purpose('Ensures that deleted SDAI-model does not belong '
      + 'to associated_models of the schema_instance to which '
      + 'this SDAI-model earlier was added.');
   bool := model2 IN schema_inst.associated_models;
   assert(NOT bool);
   verdict;

   purpose('Ensures that references to instances of the deleted model '
      + 'from other models become unset.');
   p := get_attribute(inst1, 'ESTS.OMEGA.O0', VA_NSET, ?);
   verdict;

   purpose('Ensures that delete_sdai_model reports a VT_NVLD error when '
      + 'a data dictionary SDAI-model for deletion is submitted.');
   model_dict := macro_get_data_dictionary_model;
   delete_sdai_model(model_dict, VT_NVLD, ?);
   verdict;

END_PROCEDURE; -- atc_delete_SDAI_model
(*
```

Argument definitions:

model1: (input) An SDAI-model in read-write mode, based on the ESTS schema.

model2: (input) An SDAI-model in read-write mode, based on the ESTS schema and associated with a schema_instance whose native schema is the ESTS schema; this SDAI-model shall be different from **model1** and may even belong to a different repository.

6.2.23 atc_rename_SDAI_model

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *rename SDAI-model*.

EXPRESS specification:

```
*)
PROCEDURE atc_rename_SDAI_model(modl : sdai_model);
   LOCAL
      repo : sdai_repository := modl.repository;
      model_created : sdai_model := create_sdai_model(repo,
                        'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
      model_dict : sdai_model;
      inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      inst2 : application_instance := create_entity_instance('ESTS.IOTA',
         model_created, NO_ERROR, ?);
      p : primitive;
   END_LOCAL;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   atc('atc_rename_SDAI_model');

   (* Prepares an instance in one SDAI-model that will be renamed,
     to reference an instance in another SDAI-model.*)
   put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst2, ?), NO_ERROR, ?);

   purpose('Ensures that rename_sdai_model reports a VA_NSET error '
     + 'when a new name is not submitted.');
   rename_sdai_model(modl, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that rename_sdai_model reports an MO_DUP error when '
     + 'the new name submitted coincides with the name of some other '
     + 'model in the same repository.');
   rename_sdai_model(modl, 'exceptional_name_within_repo', MO_DUP, modl);
   verdict;

   purpose('Ensures that rename_sdai_model works correctly when '
     + 'parameters are correct.');
   rename_sdai_model(modl, 'another_exceptional_name_within_repo',
     NO_ERROR, ?);
   verdict;

   purpose('Ensures that references from the renamed SDAI-model to other '
     + 'SDAI-models are retained.');
   p := get_attribute(inst1, 'ESTS.OMEGA.O0', NO_ERROR, ?);
   verdict;

   purpose('Ensures that rename_sdai_model reports a VT_NVLD error '
     + 'when a data dictionary SDAI-model for renaming is submitted.');
   model_dict := macro_get_data_dictionary_model;
   rename_sdai_model(model_dict, 'exceptional_name', VT_NVLD, ?);
   verdict;

END_PROCEDURE; -- atc_rename_SDAI_model
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.24 atc_get_entity_definition

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get entity definition*.

EXPRESS specification:

```
*)
PROCEDURE atc_get_entity_definition(modl : sdai_model);
   LOCAL
     def : entity_definition;
   END_LOCAL;

   atc('atc_get_entity_definition');

   if (modl.repository.session.sdai_implementation.implementation_class
       > 1) then

     purpose('Ensures that get_entity_definition reports a VA_NSET error '
       + 'when entity name is not submitted.');
     def := get_entity_definition(modl, ?, VA_NSET, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
    verdict;

    purpose('Ensures that get_entity_definition reports an ED_NDEF '
      + 'error when entity name is not defined or declared in the '
      + 'schema which is underlying for the specified model.');
    def := get_entity_definition(modl, 'EKS', ED_NDEF, ?);
    verdict;

    purpose('Ensures that get_entity_definition works correctly when '
      + 'all parameters are correct.');
    def := get_entity_definition(modl, 'OMEGA', NO_ERROR, ?);
    verdict;
  end_if;
END_PROCEDURE; -- atc_get_entity_definition
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.25 atc_create_entity_instance

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *create entity instance*.

EXPRESS specification:

```
*)
PROCEDURE atc_create_entity_instance(repo : sdai_repository);
  LOCAL
    modl : sdai_model := create_sdai_model(repo,
           'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
    inst : application_instance;
    extent : entity_extent;
    label : STRING;
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_create_entity_instance');

  purpose('Ensures that create_entity_instance reports an ED_NDEF '
    + 'error when entity definition is not submitted.');
  inst := create_entity_instance(?, modl, ED_NDEF, ?);
  verdict;

  purpose('Ensures that create_entity_instance reports an ED_NVLD '
    + 'error when entity, instance of which is to be created, '
    + 'is not defined or declared in the schema which is underlying '
    + 'for the specified model.');
  inst := create_entity_instance('EKS', modl, ED_NVLD, 'EKS');
  verdict;

  purpose('Ensures that create_entity_instance works correctly '
    + 'when parameters are correct.');
  inst := create_entity_instance('OMEGA', modl, NO_ERROR, ?);
  verdict;

  purpose('Ensures that instance created belongs to instances set '
    + 'of the sdai_model_contents.');
  bool := inst IN modl.contents.instances;
  assert(bool);
  verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that entity extent containing created instance '
      + 'belongs to populated_folders set of the sdai_model_contents.');
   extent := macro_get_entity_extent('OMEGA');
   bool := macro_check_extent_if_populated(extent, modl);
   assert(bool);
   verdict;

   purpose('Ensures that values of attributes of the created instance '
      + 'are unset.');
   bool := macro_check_instance_if_values_unset(inst);
   assert(bool);
   verdict;

   purpose('Ensures that a persistent label for the created instance '
      + 'is unique within the repository to which this instance belongs.');
   label := get_persistent_label(inst, NO_ERROR, ?);
   delete_application_instance(inst, NO_ERROR, ?);
   inst := get_session_identifier(label, repo, EI_NEXS, ?);
   verdict;

END_PROCEDURE; -- atc_create_entity_instance
(*
```

Argument definitions:

repo: (input) An open repository (that is, from the active_servers set of the SDAI session).

6.2.26 atc_undo_changes

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *undo changes*.

EXPRESS specification:

```
*)
PROCEDURE atc_undo_changes(repo : sdai_repository);
   LOCAL
      modl : sdai_model := create_sdai_model(repo,
             'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
      inst : application_instance;
      p : primitive;
   END_LOCAL;

   atc('atc_undo_changes');

   purpose('Ensures that undo_changes reports an MX_NRW error when '
      + 'SDAI-model access is not read-write.');
   undo_changes(modl, MX_NRW, modl);
   verdict;

   (* Prepares some data for checking the undo_changes operation.*)
   start_read_write_access(modl, NO_ERROR, ?);
   inst := create_entity_instance('ESTS.OMEGA', modl, NO_ERROR, ?);
   put_attribute(inst, 'ESTS.OMEGA.O1', asp(3.4, ?), NO_ERROR, ?);
   save_changes(modl, NO_ERROR, ?);
   put_attribute(inst, 'ESTS.OMEGA.O1', asp(15.5, ?), NO_ERROR, ?);

   purpose('Ensures that undo_changes works correctly when '
      + 'SDAI-model access is read-write.');
   undo_changes(modl, NO_ERROR, ?);
   verdict;

   purpose('Ensures that the old contents of the SDAI-model was '
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      + 'restored.');
   p := get_attribute(inst, 'ESTS.OMEGA.01', NO_ERROR, ?);
   assert(p = asp(3.4, ?));
   verdict;

   purpose('Ensures that after the undo_changes operation the '
      + 'existing read-write access for the SDAI-model continues '
      + 'to be active.');
   inst := create_entity_instance('ESTS.SIGMA', modl, NO_ERROR, ?);
   verdict;

END_PROCEDURE; -- atc_undo_changes
(*
```

Argument definitions:

repo: (input) An open repository (that is, from the active_servers set of the SDAI session).

6.2.27 atc_save_changes

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *save changes*.

EXPRESS specification:

```
*)
PROCEDURE atc_save_changes(repo : sdai_repository);
   LOCAL
      modl : sdai_model := create_sdai_model(repo,
             'some_exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
      inst : application_instance;
      date : STRING;
      p : primitive;
   END_LOCAL;

   atc('atc_save_changes');

   purpose('Ensures that save_changes reports an MX_NRW error when '
      * 'SDAI-model access is not read-write.');
   save_changes(modl, MX_NRW, modl);
   verdict;

   (* An instance within the SDAI-model to be saved is created.
      Also, the value of the attribute change_date in this SDAI-model
      for future comparison is retained.*)
   start_read_write_access(modl, NO_ERROR, ?);
   inst := create_entity_instance('ESTS.OMEGA', modl, NO_ERROR, ?);
   put_attribute(inst, 'ESTS.OMEGA.01', asp(3.4, ?), NO_ERROR, ?);
   date := modl.change_date;

   purpose('Ensures that save_changes works correctly when '
      + 'SDAI-model access is read-write.');
   save_changes(modl, NO_ERROR, ?);
   verdict;

   purpose('Ensures that the attribute change_date in the '
      + 'SDAI-model considered was set to the new date.');
   assert(date <> modl.change_date);
   verdict;

   purpose('Ensures that changes in the SDAI-model indeed were saved.');
   undo_changes(modl, NO_ERROR, ?);
   p := get_attribute(inst, 'ESTS.OMEGA.01', NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   assert(p = asp(3.4, ?));
   verdict;

   purpose('Ensures that for the SDAI-model saved the existing '
      + 'read-write access continues to be active.');
   inst := create_entity_instance('ESTS.SIGMA', modl, NO_ERROR, ?);
   verdict;

END_PROCEDURE; -- atc_save_changes
(*
```

Argument definitions:

repo: (input) An open repository (that is, from the active_servers set of the SDAI session).

6.2.28 atc_get_complex_entity_definition

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get complex entity definition*.

EXPRESS specification:

```
*)
PROCEDURE atc_get_complex_entity_definition;
   LOCAL
      def : entity_definition;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_get_complex_entity_definition');

   purpose('Ensures that get_complex_entity_definition reports a '
      + 'VA_NSET error when the list of constituent simple entity '
      + 'types is either empty or not submitted at all.');
   def := get_complex_entity_definition(?, VA_NSET, ?);
   def := get_complex_entity_definition([], VA_NSET, ?);
   verdict;

   purpose('Ensures that get_complex_entity_definition reports an '
      + 'ED_NDEF error when the submitted list of constituent simple '
      + 'entity data types contains an entity that is not known '
      + 'in the data dictionary.');
   def := get_complex_entity_definition(['BETA', 'EKS'], ED_NDEF, ?);
   verdict;

   purpose('Ensures that get_complex_entity_definition works correctly '
      + 'when the list of constituent simple entity types is provided.');
   def := get_complex_entity_definition(['BETA', 'GAMMA'], NO_ERROR, ?);
   verdict;

   purpose('Ensures that the list of supertypes of the complex '
      + 'entity definition obtained contains the correct items.');
   bool := macro_compare_aggregates(def.supertypes, ['BETA', 'GAMMA']);
   assert(bool);
   verdict;

   purpose('Ensures that BOOLEAN attributes of the complex '
      + 'entity definition obtained have correct values.');
   assert(def.complex);
   assert(def.instantiable);
   assert(def.independent);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
END_PROCEDURE; -- atc_get_complex_entity_definition
(*
```

6.2.29 atc_is_subtype_of

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *is subtype of*.

EXPRESS specification:

```
*)
PROCEDURE atc_is_subtype_of;
  LOCAL
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_is_subtype_of');

  purpose('Ensures that is_subtype_of reports a ED_NDEF error when at '
    + 'least one entity definition in the pair to be checked '
    + 'is not submitted.');
  bool := is_subtype_of(?, 'ALPHA', ED_NDEF, ?);
  bool := is_subtype_of('GAMMA', ?, ED_NDEF, ?);
  verdict;

  purpose('Ensures that is_subtype_of works correctly when the '
    + 'first entity definition is a subtype of the second.');
  bool := is_subtype_of('GAMMA', 'ALPHA', NO_ERROR, ?);
  assert(bool);
  verdict;

  purpose('Ensures that is_subtype_of works correctly when the '
    + 'first entity definition is not a subtype of the second.');
  bool := is_subtype_of('GAMMA', 'BETA', NO_ERROR, ?);
  assert(NOT bool);
  verdict;

END_PROCEDURE; -- atc_is_subtype_of
(*
```

6.2.30 atg_attribute_basic

This abstract test group shall be used for the assesment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute*, *unset attribute value*, and *create aggregate instance* for attribute parameters of different types.

EXPRESS specification:

```
*)
PROCEDURE atg_attribute_basic(modl : sdai_model);

  (* The type of the attribute is NUMBER.*)
  atc_attribute_number(modl);

  (* The type of the attribute is REAL.*)
  atc_attribute_real(modl);

  (* The type of the attribute is INTEGER.*)
  atc_attribute_integer(modl);

  (* The type of the attribute is LOGICAL.*)
  atc_attribute_logical(modl);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
  (* The type of the attribute is BOOLEAN.*)
  atc_attribute_boolean(modl);

  (* The type of the attribute is STRING.*)
  atc_attribute_string(modl);

  (* The type of the attribute is BINARY.*)
  atc_attribute_binary(modl);

  (* The type of the attribute is ENUMERATION.*)
  atc_attribute_enumeration(modl);

  (* The type of the attribute is simple defined type.*)
  atc_attribute_simple_defined_type(modl);

  (* The type of the attribute is select data type and one of possible
     values may be aggregate.*)
  atc_attribute_select_aggregate_type(modl);

  (* The type of the attribute is entity.*)
  atc_attribute_entity(modl);

  (* The type of the attribute is aggregation data type.*)
  atc_attribute_aggregate(modl);

  (* Testing if attribute is submitted to the SDAI operations and is
     a correct one.*)
  atc_attribute_correctness(modl);

END_PROCEDURE;
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.31 atc_attribute_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the attribute parameter is of type NUMBER.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_number(modl : sdai_model);
  LOCAL
    inst : application_instance := create_entity_instance('ESTS.OMEGA',
      modl, NO_ERROR, ?);
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_attribute_number');

  purpose('Ensures that the attribute is initially unset after '
    + 'creation of the instance.');
  bool := test_attribute(inst, 'ESTS.OMEGA.01', NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that get_attribute reports a VA_NSET error '
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      + 'when the attribute has no value.');
   p := get_attribute(inst, 'ESTS.OMEGA.O1', VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VA_NSET error '
      + 'when an unset value for the attribute is submitted.');
   put_attribute(inst, 'ESTS.OMEGA.O1', ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VT_NVLD error '
      + 'when used for a value of a wrong type.');
   put_attribute(inst, 'ESTS.OMEGA.O1', asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_attribute works correctly when '
      + 'all parameters are correct.');
   put_attribute(inst, 'ESTS.OMEGA.O1', asp(3.4, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that test_attribute returns TRUE after '
      + 'successfully invoking put_attribute.');
   bool := test_attribute(inst, 'ESTS.OMEGA.O1', NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that get_attribute retrieves the right value.');
   p := get_attribute(inst, 'ESTS.OMEGA.O1', NO_ERROR, ?);
   assert(p = asp(3.4, ?));
   verdict;

   purpose('Ensures that the attribute value can be unset.');
   unset_attribute_value(inst, 'ESTS.OMEGA.O1', NO_ERROR, ?);
   verdict;

   purpose('Ensures that the attribute is really unset after '
      + 'unset_attribute_value operation.');
   bool := test_attribute(inst, 'ESTS.OMEGA.O1', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_number
(*
```

<u>Argument definitions:</u>

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.32 atc_attribute_real

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the attribute parameter is of type REAL.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_attribute_real(modl : sdai_model);
   LOCAL
      inst : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   atc('atc_attribute_real');

   purpose('Ensures that the attribute is initially unset '
     + 'after creation of the instance.');
   bool := test_attribute(inst, 'ESTS.OMEGA.02', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_attribute reports a VA_NSET '
     + 'error when the attribute has no value.');
   p := get_attribute(inst, 'ESTS.OMEGA.02', VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VA_NSET '
     + 'error when an unset value for the attribute is submitted.');
   put_attribute(inst, 'ESTS.OMEGA.02', ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VT_NVLD '
     + 'error when used for a value of a wrong type.');
   put_attribute(inst, 'ESTS.OMEGA.02', asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_attribute works correctly '
     + 'when all parameters are correct.');
   put_attribute(inst, 'ESTS.OMEGA.02', asp(5.6, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that test_attribute returns TRUE '
     + 'after successfully invoking put_attribute.');
   bool := test_attribute(inst, 'ESTS.OMEGA.02', NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that get_attribute retrieves the right value.');
   p := get_attribute(inst, 'ESTS.OMEGA.02', NO_ERROR, ?);
   assert(p = asp(5.6, ?));
   verdict;

   purpose('Ensures that the attribute value can be unset.');
   unset_attribute_value(inst, 'ESTS.OMEGA.02', NO_ERROR, ?);
   verdict;

   purpose('Ensures that the attribute is really unset after '
     + 'unset_attribute_value operation.');
   bool := test_attribute(inst, 'ESTS.OMEGA.02', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_real
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.33 atc_attribute_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the attribute parameter is of type INTEGER.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_integer(modl : sdai_model);
  LOCAL
    inst : application_instance := create_entity_instance('ESTS.OMEGA',
      modl, NO_ERROR, ?);
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_attribute_integer');

  purpose('Ensures that the attribute is initially unset after creation '
    + 'of the instance.');
  bool := test_attribute(inst, 'ESTS.OMEGA.O3', NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that get_attribute reports a VA_NSET error when '
    + 'the attribute has no value.');
  p := get_attribute(inst, 'ESTS.OMEGA.O3', VA_NSET, ?);
  verdict;

  purpose('Ensures that put_attribute reports a VA_NSET error when '
    + 'an unset value for the attribute is submitted.');
  put_attribute(inst, 'ESTS.OMEGA.O3', ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that put_attribute reports a VT_NVLD error when '
    + 'used for a value of a wrong type.');
  put_attribute(inst, 'ESTS.OMEGA.O3', asp('something', ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that put_attribute works correctly when all '
    + 'parameters are correct.');
  put_attribute(inst, 'ESTS.OMEGA.O3', asp(7, ?), NO_ERROR, ?);
  verdict;

  purpose('Ensures that test_attribute returns TRUE after '
    + 'successfully invoking put_attribute.');
  bool := test_attribute(inst, 'ESTS.OMEGA.O3', NO_ERROR, ?);
  assert(bool);
  verdict;

  purpose('Ensures that get_attribute retrieves the right value.');
  p := get_attribute(inst, 'ESTS.OMEGA.O3', NO_ERROR, ?);
  assert(p = asp(7, ?));
  verdict;

  purpose('Ensures that the attribute value can be unset.');
  unset_attribute_value(inst, 'ESTS.OMEGA.O3', NO_ERROR, ?);
  verdict;

  purpose('Ensures that the attribute is really unset after '
    + 'unset_attribute_value operation.');
  bool := test_attribute(inst, 'ESTS.OMEGA.O3', NO_ERROR, ?);
  assert(NOT bool);
  verdict;

END_PROCEDURE; -- atc_attribute_integer
(*
```

Argument definitions:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.34 atc_attribute_logical

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the attribute parameter is of type LOGICAL.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_logical(modl : sdai_model);
  LOCAL
    inst : application_instance := create_entity_instance('ESTS.OMEGA',
      modl, NO_ERROR, ?);
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_attribute_logical');

  purpose('Ensures that the attribute is initially unset after creation '
    + 'of the instance.');
  bool := test_attribute(inst, 'ESTS.OMEGA.04', NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that get_attribute reports a VA_NSET error when '
    + 'the attribute has no value.');
  p := get_attribute(inst, 'ESTS.OMEGA.04', VA_NSET, ?);
  verdict;

  purpose('Ensures that put_attribute reports a VA_NSET error '
    + 'when an unset value for the attribute is submitted.');
  put_attribute(inst, 'ESTS.OMEGA.04', ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that put_attribute reports a VT_NVLD error '
    + 'when used for a value of a wrong type.');
  put_attribute(inst, 'ESTS.OMEGA.04', asp('something', ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that put_attribute works correctly when all '
    + 'parameters are correct.');
  put_attribute(inst, 'ESTS.OMEGA.04', asp(UNKNOWN, ?), NO_ERROR, ?);
  verdict;

  purpose('Ensures that test_attribute returns TRUE after '
    + 'successfully invoking put_attribute.');
  bool := test_attribute(inst, 'ESTS.OMEGA.04', NO_ERROR, ?);
  assert(bool);
  verdict;

  purpose('Ensures that get_attribute retrieves the right value.');
  p := get_attribute(inst, 'ESTS.OMEGA.04', NO_ERROR, ?);
  assert(p = asp(UNKNOWN, ?));
  verdict;

  purpose('Ensures that the attribute value can be unset.');
  unset_attribute_value(inst, 'ESTS.OMEGA.04', NO_ERROR, ?);
  verdict;

  purpose('Ensures that the attribute is really unset after '
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      + 'unset_attribute_value operation.');
   bool := test_attribute(inst, 'ESTS.OMEGA.04', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_logical
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.35 atc_attribute_boolean

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the attribute parameter is of type BOOLEAN.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_boolean(modl : sdai_model);
   LOCAL
      inst : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_attribute_boolean');

   purpose('Ensures that the attribute is initially unset after creation '
      + 'of the instance.');
   bool := test_attribute(inst, 'ESTS.OMEGA.05', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_attribute reports a VA_NSET error '
      + 'when the attribute has no value.');
   p := get_attribute(inst, 'ESTS.OMEGA.05', VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VA_NSET error '
      + 'when an unset value for the attribute is submitted.');
   put_attribute(inst, 'ESTS.OMEGA.05', ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VT_NVLD error '
      + 'when used for a value of a wrong type.');
   put_attribute(inst, 'ESTS.OMEGA.05', asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_attribute works correctly when all '
      + 'parameters are correct.');
   put_attribute(inst, 'ESTS.OMEGA.05', asp(FALSE, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that test_attribute returns TRUE after '
      + 'successfully invoking put_attribute.');
   bool := test_attribute(inst, 'ESTS.OMEGA.05', NO_ERROR, ?);
   assert(bool);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   purpose('Ensures that get_attribute retrieves the right value.');
   p := get_attribute(inst, 'ESTS.OMEGA.05', NO_ERROR, ?);
   assert(p = asp(FALSE, ?));
   verdict;

   purpose('Ensures that the attribute value can be unset.');
   unset_attribute_value(inst, 'ESTS.OMEGA.05', NO_ERROR, ?);
   verdict;

   purpose('Ensures that the attribute is really unset after '
      + 'unset_attribute_value operation.');
   bool := test_attribute(inst, 'ESTS.OMEGA.05', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_boolean
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.36 atc_attribute_string

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the attribute parameter is of type STRING.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_string(modl : sdai_model);
   LOCAL
      inst : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_attribute_string');

   purpose('Ensures that the attribute is initially unset after '
      + 'creation of the instance.');
   bool := test_attribute(inst, 'ESTS.OMEGA.06', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_attribute reports a VA_NSET error '
      + 'when the attribute has no value.');
   p := get_attribute(inst, 'ESTS.OMEGA.06', VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VA_NSET error '
      + 'when an unset value for the attribute is submitted.');
   put_attribute(inst, 'ESTS.OMEGA.06', ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VT_NVLD error '
      + 'when used for a value of a wrong type.');
   put_attribute(inst, 'ESTS.OMEGA.06', asp(7.7, ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_attribute works correctly when '
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      + 'all parameters are correct.');
   put_attribute(inst, 'ESTS.OMEGA.06', asp('test-string', ?),
      NO_ERROR, ?);
   verdict;

   purpose('Ensures that test_attribute returns TRUE after '
      + 'successfully invoking put_attribute.');
   bool := test_attribute(inst, 'ESTS.OMEGA.06', NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that get_attribute retrieves the right value.');
   p := get_attribute(inst, 'ESTS.OMEGA.06', NO_ERROR, ?);
   assert(p = asp('test-string', ?));
   verdict;

   purpose('Ensures that the attribute value can be unset.');
   unset_attribute_value(inst, 'ESTS.OMEGA.06', NO_ERROR, ?);
   verdict;

   purpose('Ensures that the attribute is really unset after '
      + 'unset_attribute_value operation.');
   bool := test_attribute(inst, 'ESTS.OMEGA.06', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_string
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.37 atc_attribute_binary

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the attribute parameter is of type BINARY.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_binary(modl : sdai_model);
   LOCAL
      inst : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_attribute_binary');

   purpose('Ensures that the attribute is initially unset after '
      + 'creation of the instance.');
   bool := test_attribute(inst, 'ESTS.OMEGA.07', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_attribute reports a VA_NSET error '
      + 'when the attribute has no value.');
   p := get_attribute(inst, 'ESTS.OMEGA.07', VA_NSET, ?);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   purpose('Ensures that put_attribute reports a VA_NSET error '
      + 'when an unset value for the attribute is submitted.');
   put_attribute(inst, 'ESTS.OMEGA.07', ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VT_NVLD error '
      + 'when used for a value of a wrong type.');
   put_attribute(inst, 'ESTS.OMEGA.07', asp(7.7, ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_attribute works correctly when '
      + 'all parameters are correct.');
   put_attribute(inst, 'ESTS.OMEGA.07', asp(%111011, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that test_attribute returns TRUE after '
      + 'successfully invoking put_attribute.');
   bool := test_attribute(inst, 'ESTS.OMEGA.07', NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that get_attribute retrieves the right value.');
   p := get_attribute(inst, 'ESTS.OMEGA.07', NO_ERROR, ?);
   assert(p = asp(%111011, ?));
   verdict;

   purpose('Ensures that the attribute value can be unset.');
   unset_attribute_value(inst, 'ESTS.OMEGA.07', NO_ERROR, ?);
   verdict;

   purpose('Ensures that the attribute is really unset after '
      + 'unset_attribute_value operation.');
   bool := test_attribute(inst, 'ESTS.OMEGA.07', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_binary
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.38 atc_attribute_enumeration

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the attribute parameter is of an enumeration data type.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_enumeration(modl : sdai_model);
   LOCAL
      inst : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_attribute_enumeration');

   purpose('Ensures that the attribute is initially unset after '
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
     + 'creation of the instance.');
   bool := test_attribute(inst, 'ESTS.OMEGA.08', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_attribute reports a VA_NSET error '
     + 'when the attribute has no value.');
   p := get_attribute(inst, 'ESTS.OMEGA.08', VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VA_NSET error '
     + 'when an unset value for the attribute is submitted.');
   put_attribute(inst, 'ESTS.OMEGA.08', ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VT_NVLD error '
     + 'when used for a value of a wrong type.');
   put_attribute(inst, 'ESTS.OMEGA.08', asp(7.7, ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_attribute works correctly when '
     + 'all parameters are correct.');
   put_attribute(inst, 'ESTS.OMEGA.08', asp(tau.stigma, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that test_attribute returns TRUE after '
     + 'successfully invoking put_attribute.');
   bool := test_attribute(inst, 'ESTS.OMEGA.08', NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that get_attribute retrieves the right value.');
   p := get_attribute(inst, 'ESTS.OMEGA.08', NO_ERROR, ?);
   assert(p :=: asp(tau.stigma, ?));
   verdict;

   purpose('Ensures that the attribute value can be unset.');
   unset_attribute_value(inst, 'ESTS.OMEGA.08', NO_ERROR, ?);
   verdict;

   purpose('Ensures that the attribute is really unset after '
     + 'unset_attribute_value operation.');
   bool := test_attribute(inst, 'ESTS.OMEGA.08', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_enumeration
(*
```

<u>Argument definitions:</u>

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.39 atc_attribute_simple_defined_type

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the attribute parameter is of type *defined type*.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_attribute_simple_defined_type(modl : sdai_model);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   LOCAL
      inst : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_attribute_simple_defined_type');

   purpose('Ensures that the attribute is initially unset after '
      + 'creation of the instance.');
   bool := test_attribute(inst, 'ESTS.OMEGA.09', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_attribute reports a VA_NSET error '
      + 'when the attribute has no value.');
   p := get_attribute(inst, 'ESTS.OMEGA.09', VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VA_NSET error '
      + 'when an unset value for the attribute is submitted.');
   put_attribute(inst, 'ESTS.OMEGA.09', ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VT_NVLD error '
      + 'when used for a value of a wrong type.');
   put_attribute(inst, 'ESTS.OMEGA.09', asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_attribute works correctly when '
      + 'all parameters are correct.');
   put_attribute(inst, 'ESTS.OMEGA.09', asp(89, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that test_attribute returns TRUE '
      + 'after successfully invoking put_attribute.');
   bool := test_attribute(inst, 'ESTS.OMEGA.09', NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that get_attribute retrieves the right value.');
   p := get_attribute(inst, 'ESTS.OMEGA.09', NO_ERROR, ?);
   assert(p = asp(89, ?));
   verdict;

   purpose('Ensures that the attribute value can be unset.');
   unset_attribute_value(inst, 'ESTS.OMEGA.09', NO_ERROR, ?);
   verdict;

   purpose('Ensures that the attribute is really unset after '
      + 'unset_attribute_value operation.');
   bool := test_attribute(inst, 'ESTS.OMEGA.09', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_simple_defined_type
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.40 atc_attribute_select_aggregate_type

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *create aggregate instance* and *unset attribute value* for the case that the attribute parameter is of a *select data type* whose set of selectable types includes *aggregation type*.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_select_aggregate_type(modl : sdai_model);
  LOCAL
    inst : application_instance := create_entity_instance('ESTS.EPSILON',
      modl, NO_ERROR, ?);
    aggr1 : aggregate_primitive; -- SET [0:3] OF nu
    aggr2 : aggregate_primitive; -- SET [0:3] OF nu
    aggr3 : aggregate_primitive; -- LIST [1:3] OF REAL
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_attribute_select_aggregate_type');

  purpose('Ensures that the attribute is initially unset after '
    + 'creation of the instance.');
  bool := test_attribute(inst, 'ESTS.EPSILON.E5', NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that get_attribute reports a VA_NSET error '
    + 'when the attribute has no value.');
  p := get_attribute(inst, 'ESTS.EPSILON.E5', VA_NSET, ?);
  verdict;

  purpose('Ensures that create_aggregate_instance works correctly '
    + 'when parameters are correct.');
  aggr1 := create_aggregate_instance(inst, 'ESTS.EPSILON.E5',
    ['ESTS.UPSILON'], NO_ERROR, ?);
  verdict;

  purpose('Ensures that test_attribute returns TRUE '
    + 'after successfully invoking create_aggregate_instance.');
  bool := test_attribute(inst, 'ESTS.EPSILON.E5', NO_ERROR, ?);
  assert(bool);
  verdict;

  purpose('Ensures that the aggregate created can contain '
    + 'elements of different data types.');
  add_unordered(aggr1, asp(10, ['ESTS.XI']), NO_ERROR, ?);
  aggr3 := create_aggregate_instance_unordered(aggr1, ['ESTS.CHI'],
    NO_ERROR, ?);
  verdict;

  purpose('Ensures that get_attribute retrieves the right value.');
  p := get_attribute(inst, 'ESTS.EPSILON.E5', NO_ERROR, ?);
  aggr2 := macro_convert_primitive_to_aggregate(p);
  assert(aggr1 :=: aggr2);
  verdict;

  purpose('Ensures that the attribute value can be unset.');
  unset_attribute_value(inst, 'ESTS.EPSILON.E5', NO_ERROR, ?);
  verdict;

  purpose('Ensures that the attribute is really unset after '
    + 'unset_attribute_value operation.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   bool := test_attribute(inst, 'ESTS.EPSILON.E5', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_select_aggregate_type
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.41 atc_attribute_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute* and *unset attribute value* for the case that the type of the attribute parameter is an entity data type.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_entity(modl : sdai_model);
   LOCAL
     inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
       modl, NO_ERROR, ?);
     inst2 : application_instance;
     bool : BOOLEAN;
     p : primitive;
   END_LOCAL;

   atc('atc_attribute_entity');

   purpose('Ensures that the attribute is initially unset after '
     + 'creation of the instance.');
   bool := test_attribute(inst1, 'ESTS.OMEGA.O0', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_attribute reports a VA_NSET error '
     + 'when the attribute has no value.');
   p := get_attribute(inst1, 'ESTS.OMEGA.O0', VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VA_NSET error '
     + 'when an unset value for the attribute is submitted.');
   put_attribute(inst1, 'ESTS.OMEGA.O0', ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_attribute reports a VT_NVLD error '
     + 'when used for a value of a wrong type.');
   inst2 := create_entity_instance('ESTS.EPSILON', modl, NO_ERROR, ?);
   put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst2, ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_attribute works correctly when '
     + 'all parameters are correct.');
   inst2 := create_entity_instance('ESTS.IOTA', modl, NO_ERROR, ?);
   put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst2, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that test_attribute returns TRUE after '
     + 'successfully invoking put_attribute.');
   bool := test_attribute(inst1, 'ESTS.OMEGA.O0', NO_ERROR, ?);
   assert(bool);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   verdict;

   purpose('Ensures that get_attribute retrieves the right value.');
   p := get_attribute(inst1, 'ESTS.OMEGA.O0', NO_ERROR, ?);
   assert(p :=: asp(inst2, ?));
   verdict;

   purpose('Ensures that the attribute value can be unset.');
   unset_attribute_value(inst1, 'ESTS.OMEGA.O0', NO_ERROR, ?);
   verdict;

   purpose('Ensures that the attribute is really unset after '
      + 'unset_attribute_value operation.');
   bool := test_attribute(inst1, 'ESTS.OMEGA.O0', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_attribute_entity
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.42 atc_attribute_aggregate

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *create aggregate instance* and *unset attribute value* for the case that the type of the attribute parameter is an *aggregation data type*.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_aggregate(modl : sdai_model);
   LOCAL
      inst : application_instance := create_entity_instance('ESTS.EPSILON',
         modl, NO_ERROR, ?);
      aggr1, aggr2 : aggregate_instance; -- LIST [1:?] OF nu
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_attribute_aggregate');

   purpose('Ensures that the attribute is initially unset after '
      + 'creation of the instance.');
   bool := test_attribute(inst, 'ESTS.EPSILON.E2', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_attribute reports a VA_NSET error '
      + 'when the attribute has no value.');
   p := get_attribute(inst, 'ESTS.EPSILON.E2', VA_NSET, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance works correctly '
      + 'when parameters are correct.');
   aggr1 := create_aggregate_instance(inst, 'ESTS.EPSILON.E2', ?,
      NO_ERROR, ?);
   verdict;

   purpose('Ensures that test_attribute returns TRUE after '
      + 'successfully invoking create_aggregate_instance.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
  bool := test_attribute(inst, 'ESTS.EPSILON.E2', NO_ERROR, ?);
  assert(bool);
  verdict;

  purpose('Ensures that get_attribute retrieves the right value.');
  p := get_attribute(inst, 'ESTS.EPSILON.E2', NO_ERROR, ?);
  aggr2 := macro_convert_primitive_to_aggregate(p);
  add_by_index(aggr1, 1, asp(100, ['ESTS.XI']), NO_ERROR, ?);
  assert(asp(100, ['ESTS.XI']) IN aggr2);
  verdict;

  purpose('Ensures that put_attribute cannot be applied to assign '
    + 'a value of type aggregate.');
  put_attribute(inst, 'ESTS.EPSILON.E2', asp(aggr2, ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that the attribute value can be unset.');
  unset_attribute_value(inst, 'ESTS.EPSILON.E2', NO_ERROR, ?);
  verdict;

  purpose('Ensures that the attribute is really unset after '
    + 'unset_attribute_value operation.');
  bool := test_attribute(inst, 'ESTS.EPSILON.E2', NO_ERROR, ?);
  assert(NOT bool);
  verdict;

END_PROCEDURE; -- atc_attribute_aggregate
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.43 atc_attribute_correctness

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *test attribute*, *get attribute*, *put attribute*, *create aggregate instance* and *unset attribute value*. The correctness of the attribute parameter is checked by this abstract test case. This test needs not to be performed for early binding implementations where the attribute parameter is embedded in the function or subroutine name.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_correctness(modl : sdai_model);
  LOCAL
    inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
      modl, NO_ERROR, ?);
    inst2 : application_instance := create_entity_instance(
      'ESTS.EPSILON', modl, NO_ERROR, ?);
    aggr : aggregate_instance ; -- LIST [1:?] OF nu
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_attribute_correctness');

  purpose('Ensures that put_attribute reports an AT_NDEF error when '
    + 'the attribute parameter is not submitted.');
  put_attribute(inst1, ?, asp('test-string', ?), AT_NDEF, ?);
  verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that put_attribute reports an AT_NVLD error '
     + 'when an attribute submitted is not from the type of '
     + 'the entity instance specified.');
   put_attribute(inst1, 'ESTS.EPSILON.E1', asp('test-string', ?), AT_NVLD,
     'ESTS.EPSILON.E1');
   verdict;

   purpose('Ensures that test_attribute reports an AT_NDEF error '
     + 'when attribute is not submitted.');
   bool := test_attribute(inst1, ?, AT_NDEF, ?);
   verdict;

   purpose('Ensures that test_attribute reports an AT_NVLD error '
     + 'when an attribute submitted is not from the type of the '
     + 'entity instance specified.');
   bool := test_attribute(inst1, 'ESTS.EPSILON.E1', AT_NVLD,
     'ESTS.EPSILON.E1');
   verdict;

   purpose('Ensures that get_attribute reports an AT_NDEF error '
     + 'when attribute is not submitted.');
   p := get_attribute(inst1, ?, AT_NDEF, ?);
   verdict;

   purpose('Ensures that get_attribute reports an AT_NVLD error '
     + 'when an attribute submitted is not from the type of '
     + 'the entity instance specified.');
   p := get_attribute(inst1, 'ESTS.EPSILON.E1', AT_NVLD,
     'ESTS.EPSILON.E1');
   verdict;

   purpose('Ensures that unset_attribute_value reports an '
     + 'AT_NDEF error when attribute is not submitted.');
   unset_attribute_value(inst1, ?, AT_NDEF, ?);
   verdict;

   purpose('Ensures that unset_attribute_value reports an '
     + 'AT_NVLD error when an attribute submitted is not from '
     + 'the type of the entity instance specified.');
   unset_attribute_value(inst1, 'ESTS.EPSILON.E1', AT_NVLD,
     'ESTS.EPSILON.E1');
   verdict;

   purpose('Ensures that create_aggregate_instance reports an '
     + 'AT_NDEF error when attribute is not submitted.');
   aggr := create_aggregate_instance(inst2, ?, ?, AT_NDEF, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance reports an '
     + 'AT_NVLD error when an attribute submitted is not from '
     + 'the type of the entity instance specified.');
   aggr := create_aggregate_instance(inst2, 'ESTS.OMEGA.O2', ?, AT_NVLD,
     'ESTS.OMEGA.O2');
   verdict;

   purpose('Ensures that create_aggregate_instance reports an '
     + 'AT_NVLD error when the type of the attribute is not an '
     + 'aggregation data type.');
   aggr := create_aggregate_instance(inst1, 'ESTS.OMEGA.O1', ?, AT_NVLD,
     'ESTS.OMEGA.O1');
   verdict;

END_PROCEDURE; -- atc_attribute_correctness
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.44 atc_attribute_entity_instance_in_another_model

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get attribute* in the case when the value of the attribute is an entity instance that belongs to an SDAI-model in an open repository whose access is ended.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_entity_instance_in_another_model
     (model1 : sdai_model; model2 : sdai_model);
   LOCAL
     transaction : sdai_transaction;
     (* Creation of the referencing instance. *)
     inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
       model1, NO_ERROR, ?);
     (* Creation of the referenced instance. *)
     inst2 : application_instance := create_entity_instance('ESTS.IOTA',
       model2, NO_ERROR, ?);
     p : primitive;
   END_LOCAL;

   atc('atc_attribute_entity_instance_in_another_model');

   (* Setting a reference. *)
   put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst2, ?), NO_ERROR, ?);

   if (model1.repository.session.sdai_implementation.transaction_level
       = 3) then
     (* Commit operation settles all changes done; it is needed before
        ending access of the SDAI-model containing the referenced instance.
     *)
     transaction := model1.repository.session.active_transaction[1];
     commit(transaction, NO_ERROR, ?);
   end_if;

   (* A read-write access of the SDAI-model containing the referenced
      instance is ended. *)
   end_read_write_access(model2, NO_ERROR, ?);

   purpose('Ensures that get_attribute retrieves the right value.');
   p := get_attribute(inst1, 'ESTS.OMEGA.O0', NO_ERROR, ?);
   assert(p :=: asp(inst2, ?));
   verdict;

   purpose('Ensures that the read-only access of the closed model that is '
     + 'owning for the instance being referenced from another model is '
     + 'automatically started.');
   assert(model2.mode = access_type.read_only);
   verdict;

END_PROCEDURE; -- atc_attribute_entity_instance_in_another_model
(*
```

Argument definitions:

model1: (input) An SDAI-model in read-write mode, based on the ESTS schema.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

model2: (input) An SDAI-model in read-write mode, based on the ESTS schema. This SDAI-model shall be different from model1 and may even belong to a different repository.

6.2.45 atc_attribute_entity_instance_in_closed_repository

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get attribute* in the case when the value of the attribute is an entity instance within an SDAI-model in a closed repository.

EXPRESS specification:

```
*)
PROCEDURE atc_attribute_entity_instance_in_closed_repository
      (model1 : sdai_model; model2 : sdai_model);
  LOCAL
    repo1 : sdai_repository := model1.repository;
    repo2 : sdai_repository := model2.repository;
    transaction : sdai_transaction;
    (* Creation of the referencing instance. *)
    inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
      model1, NO_ERROR, ?);
    (* Creation of the referenced instance. *)
    inst2 : application_instance := create_entity_instance('ESTS.IOTA',
      model2, NO_ERROR, ?);
    p : primitive;
  END_LOCAL;

  atc('atc_attribute_entity_instance_in_closed_repository');

  (* Setting a reference. *)
  put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst2, ?), NO_ERROR, ?);

  if (repo1.session.sdai_implementation.transaction_level = 3) then
    (* Commit operation settles all changes done; it is needed before
       closing repository containing the referenced instance. *)
    transaction := repo1.session.active_transaction[1];
    commit(transaction, NO_ERROR, ?);
  end_if;

  (* Repository containing the referenced instance is closed. *)
  close_repository(repo2, NO_ERROR, ?);

  purpose('Ensures that get_attribute reports an RP_NOPN error when the '
    + 'repository containing the instance referenced by the '
    + 'specified attribute is closed.');
  p := get_attribute(inst1, 'ESTS.OMEGA.O0', RP_NOPN, repo2);
  verdict;

END_PROCEDURE; -- atc_attribute_entity_instance_in_closed_repository
(*
```

Argument definitions:

model1: (input) An SDAI-model in read-write mode, based on the ESTS schema.

model2: (input) An SDAI-model in read-write mode, based on the ESTS schema. This SDAI-model shall belong to a different repository than model1 does.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.46 atc_find_entity_instance_sdai_model

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *find entity instance SDAI-model*.

EXPRESS specification:

```
*)
PROCEDURE atc_find_entity_instance_sdai_model(modl : sdai_model);
  LOCAL
    model_found : sdai_model;
    inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
      modl, NO_ERROR, ?);
    inst2 : application_instance;
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_find_entity_instance_sdai_model');

  purpose('Ensures that the model found is the same as that in '
    + 'which the instance submitted for '
    + 'find_entity_instance_sdai_model was created.');
  model_found := find_entity_instance_sdai_model(inst1, NO_ERROR, ?);
  inst2 := create_entity_instance('ESTS.EPSILON', model_found,
    NO_ERROR, ?);
  bool := inst2 IN modl.contents.instances;
  assert(bool);
  verdict;

END_PROCEDURE; -- atc_find_entity_instance_sdai_model
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.47 atc_get_instance_type

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get instance type*.

EXPRESS specification:

```
*)
PROCEDURE atc_get_instance_type(modl : sdai_model);
  LOCAL
    def1 : entity_definition := get_entity_definition(modl, 'OMEGA',
           NO_ERROR, ?);
    def2 : entity_definition;
    inst : application_instance := create_entity_instance(def1, modl,
      NO_ERROR, ?);
  END_LOCAL;

  if (modl.repository.session.sdai_implementation.implementation_class
      > 1) then
    atc('atc_get_instance_type');

    purpose('Ensures that the correct entity definition of the '
      + 'instance is returned.');
    def2 := get_instance_type(inst, NO_ERROR, ?);
    assert(def1 :=: def2);
    verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   end_if;
END_PROCEDURE; -- atc_get_instance_type
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.48 atc_is_instance_of

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *is instance of*.

EXPRESS specification:

```
*)
PROCEDURE atc_is_instance_of(modl : sdai_model);
   LOCAL
      def : entity_definition := get_entity_definition(modl, 'OMEGA',
            NO_ERROR, ?);
      inst : application_instance := create_entity_instance(def, modl,
         NO_ERROR, ?);
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_is_instance_of');

   purpose('Ensures that is_instance_of reports an ED_NDEF error '
      + 'when entity definition is not submitted.');
   bool := is_instance_of(inst, ?, ED_NDEF, ?);
   verdict;

   purpose('Ensures that is_instance_of returns TRUE when the '
      + 'instance checked is of the specified entity type.');
   bool := is_instance_of(inst, def, NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that is_instance_of returns FALSE when the '
      + 'instance checked is of entity type that is different '
      + 'than the specified one.');
   def := get_entity_definition(modl, 'EPSILON', NO_ERROR, ?);
   bool := is_instance_of(inst, def, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_is_instance_of
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.49 atc_is_kind_of

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *is kind of*.

EXPRESS specification:

```
*)
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
PROCEDURE atc_is_kind_of(modl : sdai_model);
  LOCAL
    def : entity_definition;
    inst : application_instance := create_entity_instance('ESTS.DELTA',
      modl, NO_ERROR, ?);
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_is_kind_of');

  purpose('Ensures that is_kind_of reports an ED_NDEF error when '
    + 'entity definition is not submitted.');
  bool := is_kind_of(inst, ?, ED_NDEF, ?);
  verdict;

  purpose('Ensures that is_kind_of returns TRUE when the instance '
    + 'checked is of entity type that is a subtype of the '
    + 'specified entity data type.');
  def := get_entity_definition(modl, 'ALPHA', NO_ERROR, ?);
  bool := is_kind_of(inst, def, NO_ERROR, ?);
  assert(bool);
  verdict;

  purpose('Ensures that is_instance_of returns FALSE when the '
    + 'instance checked is of entity type that is not a subtype '
    + 'of the specified entity data type.');
  def := get_entity_definition(modl, 'LAMDA', NO_ERROR, ?);
  bool := is_kind_of(inst, def, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

END_PROCEDURE; -- atc_is_kind_of
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.50 atc_persistent_label_and_session_identifier

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *get persistent label* and *get session identifier*.

EXPRESS specification:

```
*)
PROCEDURE atc_persistent_label_and_session_identifier(modl : sdai_model);
  LOCAL
    repo : sdai_repository := modl.repository;
    inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
      modl, NO_ERROR, ?);
    inst2 : application_instance;
    label : STRING;
  END_LOCAL;

  atc('atc_persistent_label_and_session_identifier');

  purpose('Ensures that get_persistent_label performs correctly when '
    + 'an instance is specified.');
  label := get_persistent_label(inst1, NO_ERROR, ?);
  verdict;

  purpose('Ensures that the label returned by get_persistent_label '
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      + 'is unique within a repository.');
   inst2 := get_session_identifier(label, repo, NO_ERROR, ?);
   assert(inst2 :=: inst1);
   verdict;

   purpose('Ensures that get_session_identifier reports an VA_NSET '
      + 'error when label is not submitted.');
   inst2 := get_session_identifier(?, repo, VA_NSET, ?);
   verdict;

   purpose('Ensures that get_session_identifier reports an EI_NEXS '
      + 'error when an instance requested is not found in the repository.');
   delete_application_instance(inst1, NO_ERROR, ?);
   inst2 := get_session_identifier(label, repo, EI_NEXS, ?);
   verdict;

   purpose('Ensures that get_session_identifier reports an RP_NOPN '
      + 'error when an instance requested belongs to the '
      + 'closed repository.');
   inst1 := create_entity_instance('ESTS.EPSILON', modl, NO_ERROR, ?);
   label := get_persistent_label(inst1, NO_ERROR, ?);
   close_repository(repo, NO_ERROR, ?);
   inst2 := get_session_identifier(label, repo, RP_NOPN, repo);
   verdict;

END_PROCEDURE; -- atc_persistent_label_and_session_identifier
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.51 atc_find_entity_instance_users

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *find entity instance users.*

EXPRESS specification:

```
*)
PROCEDURE atc_find_entity_instance_users(modl : sdai_model);
   LOCAL
      schema_inst : schema_instance := modl.associated_with[1];
      inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      inst2 : application_instance := create_entity_instance(
         'ESTS.EPSILON', modl, NO_ERROR, ?);
      inst3 : application_instance := create_entity_instance(
         'ESTS.EPSILON', modl, NO_ERROR, ?);
      inst4 : application_instance := create_entity_instance('ESTS.IOTA',
         modl, NO_ERROR, ?);
      aggr : aggregate_instance; -- LIST [1:?] OF nu
      non_persist_list_schemas : non_persistent_list_instance :=
                                 create_non_persistent_list(NO_ERROR, ?);
      non_persist_list_instances : non_persistent_list_instance :=
                                 create_non_persistent_list(NO_ERROR, ?);
      count : INTEGER;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_find_entity_instance_users');

   (* Instance inst2 references instance inst1. *)
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
put_attribute(inst2, 'ESTS.EPSILON.E1', asp(inst1, ?), NO_ERROR, ?);

(* Instance inst3 references instance inst1 (through the aggregate and
   through the attribute of entity type). *)
aggr := create_aggregate_instance(inst3, 'ESTS.EPSILON.E2', ?,
  NO_ERROR, ?);
add_by_index(aggr, 1, asp(inst1, ?), NO_ERROR, ?);
add_by_index(aggr, 2, asp(inst1, ?), NO_ERROR, ?);
put_attribute(inst3, 'ESTS.EPSILON.E5', asp(inst1, ?), NO_ERROR, ?);

purpose('Ensures that find_entity_instance_users reports an '
  + 'AI_NEXS error when a non-persistent list for writing the '
  + 'result is not provided.');
find_entity_instance_users(inst1, non_persist_list_schemas, ?,
  AI_NEXS, ?);
verdict;

purpose('Ensures that find_entity_instance_users reports an '
  + 'AI_NVLD error when an aggregate different than non-persistent '
  + 'list for writing the result is submitted.');
find_entity_instance_users(inst1, non_persist_list_schemas, aggr,
  AI_NVLD, aggr);
verdict;

purpose('Ensures that find_entity_instance_users reports an '
  + 'AI_NEXS error when a non-persistent list specifying the '
  + 'domain of entity instances is not submitted.');
find_entity_instance_users(inst1, ?, non_persist_list_instances,
  AI_NEXS, ?);
verdict;

purpose('Ensures that find_entity_instance_users performs correctly '
  + 'when all parameters are correct.');
find_entity_instance_users(inst1, non_persist_list_schemas,
  non_persist_list_instances, NO_ERROR, ?);
verdict;

purpose('Ensures that the resulting aggregate is empty (because a '
  + 'non-persistent list specifying the domain of entity instances '
  + 'is empty).');
count := get_member_count(non_persist_list_instances, NO_ERROR, ?);
assert(count = 0);
verdict;

(* Making the non-persistent list specifying the domain of entity
   instances to contain a given schema instance. *)
add_by_index(non_persist_list_schemas, 1, asp(schema_inst, ?),
  NO_ERROR, ?);

purpose('Ensures that find_entity_instance_users performs correctly '
  + 'and writes the instances referencing the specified instance into '
  + 'a submitted aggregate.');
find_entity_instance_users(inst1, non_persist_list_schemas,
  non_persist_list_instances, NO_ERROR, ?);
bool := macro_compare_aggregates(non_persist_list_instances,
  [inst2, inst3, inst3, inst3]);
assert(bool);
verdict;

(* Instance inst1 references instance inst4. *)
put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst4, ?), NO_ERROR, ?);

purpose('Ensures that find_entity_instance_users performs correctly '
  + 'when a non-persistent list submitted for apending instances '
  + 'is nonempty.');
find_entity_instance_users(inst4, non_persist_list_schemas,
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      non_persist_list_instances, NO_ERROR, ?);
   verdict;

   purpose('Ensures that the instance referencing the specified instance '
      + 'is added to a non-persistent list submitted for writing the '
      + 'result and all instances earlier included into it are retained.');
   bool := is_member(non_persist_list_instances, asp(inst1, ?),
      NO_ERROR, ?);
   assert(bool);
   count := get_member_count(non_persist_list_instances, NO_ERROR, ?);
   assert(count = 5);
   verdict;

END_PROCEDURE; -- atc_find_entity_instance_users
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema and associated with a schema instance whose native schema is ESTS schema.

6.2.52 atc_find_entity_instance_usedin

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *find entity instance usedin*.

EXPRESS specification:

```
*)
PROCEDURE atc_find_entity_instance_usedin(modl : sdai_model);
   LOCAL
      schema_inst : schema_instance := modl.associated_with[1];
      inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      inst2 : application_instance := create_entity_instance(
         'ESTS.EPSILON', modl, NO_ERROR, ?);
      inst3 : application_instance := create_entity_instance(
         'ESTS.EPSILON', modl, NO_ERROR, ?);
      inst4 : application_instance := create_entity_instance('ESTS.IOTA',
         modl, NO_ERROR, ?);
      aggr1, aggr2 : aggregate_instance; -- LIST [1:?] OF nu
      non_persist_list_schemas : non_persistent_list_instance :=
                                 create_non_persistent_list(NO_ERROR, ?);
      non_persist_list_instances : non_persistent_list_instance :=
                                   create_non_persistent_list(NO_ERROR, ?);
      count : INTEGER;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_find_entity_instance_usedin');

   (* Instance inst2 references (through the aggregate) instance inst1. *)
   aggr1 := create_aggregate_instance(inst2, 'ESTS.EPSILON.E2', ?,
      NO_ERROR, ?);
   add_by_index(aggr1, 1, asp(inst1, ?), NO_ERROR, ?);

   (* Instance inst3 references instance inst1 (through the aggregate and
      through the attribute of entity type). *)
   aggr2 := create_aggregate_instance(inst3, 'ESTS.EPSILON.E2', ?,
      NO_ERROR, ?);
   add_by_index(aggr2, 1, asp(inst1, ?), NO_ERROR, ?);
   add_by_index(aggr2, 2, asp(inst1, ?), NO_ERROR, ?);
   put_attribute(inst3, 'ESTS.EPSILON.E5', asp(inst1, ?), NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
purpose('Ensures that find_entity_instance_usedin reports an '
   + 'AI_NEXS error when a non-persistent list for writing the '
   + 'result is not provided.');
find_entity_instance_usedin(inst1, 'ESTS.EPSILON.E2',
   non_persist_list_schemas, ?, AI_NEXS, ?);
verdict;

purpose('Ensures that find_entity_instance_usedin reports an '
   + 'AI_NVLD error when an aggregate different than non-persistent '
   + 'list for writing the result is submitted.');
find_entity_instance_usedin(inst1, 'ESTS.EPSILON.E2',
   non_persist_list_schemas, aggr1, AI_NVLD, aggr1);
verdict;

purpose('Ensures that find_entity_instance_usedin reports an '
   + 'AI_NEXS error when a non-persistent list specifying the '
   + 'domain of entity instances is not submitted.');
find_entity_instance_usedin(inst1, 'ESTS.EPSILON.E2', ?,
   non_persist_list_instances, AI_NEXS, ?);
verdict;

purpose('Ensures that find_entity_instance_usedin performs '
   + 'correctly when all parameters are correct.');
find_entity_instance_usedin(inst1, 'ESTS.EPSILON.E2',
   non_persist_list_schemas, non_persist_list_instances, NO_ERROR, ?);
verdict;

purpose('Ensures that the resulting aggregate is empty (because a '
   + 'non-persistent list specifying the domain of entity instances '
   + 'is empty).');
count := get_member_count(non_persist_list_instances, NO_ERROR, ?);
assert(count = 0);
verdict;

(* Making the non-persistent list specifying the domain of entity
   instances to contain a given schema instance. *)
add_by_index(non_persist_list_schemas, 1, asp(schema_inst, ?),
   NO_ERROR, ?);

purpose('Ensures that find_entity_instance_usedin reports an '
   + 'AT_NDEF error when an attribute specifying the role '
   + 'is not submitted.');
find_entity_instance_usedin(inst1, ?, non_persist_list_schemas,
   non_persist_list_instances, AT_NDEF, ?);
verdict;

purpose('Ensures that find_entity_instance_usedin performs correctly '
   + 'and writes the instances referencing the specified instance into '
   + 'a submitted aggregate.');
find_entity_instance_usedin(inst1, 'ESTS.EPSILON.E2',
   non_persist_list_schemas, non_persist_list_instances, NO_ERROR, ?);
bool := macro_compare_aggregates(non_persist_list_instances,
   [inst2, inst3, inst3]);
assert(bool);
verdict;

(* Instance inst1 references instance inst4. *)
put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst4, ?), NO_ERROR, ?);

purpose('Ensures that find_entity_instance_usedin performs '
   + 'correctly when a non-persistent list submitted for '
   + 'apending instances is nonempty.');
find_entity_instance_usedin(inst4, 'ESTS.OMEGA.O0',
   non_persist_list_schemas, non_persist_list_instances, NO_ERROR, ?);
verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that the instance referencing the specified instance '
      + 'is added to a non-persistent list submitted for writing the '
      + 'result and all instances earlier included into it are retained.');
   bool := is_member(non_persist_list_instances, asp(inst1, ?),
      NO_ERROR, ?);
   assert(bool);
   count := get_member_count(non_persist_list_instances, NO_ERROR, ?);
   assert(count = 4);
   verdict;

END_PROCEDURE; -- atc_find_entity_instance_usedin
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema and associated with a schema instance whose native schema is ESTS schema.

6.2.53 atc_find_instance_roles

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *find instance roles.*

EXPRESS specification:

```
*)
PROCEDURE atc_find_instance_roles(modl : sdai_model);
   LOCAL
      schema_inst : schema_instance := modl.associated_with[1];
      inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
         modl, NO_ERROR, ?);
      inst2 : application_instance := create_entity_instance(
         'ESTS.EPSILON', modl, NO_ERROR, ?);
      inst3 : application_instance := create_entity_instance(
         'ESTS.EPSILON', modl, NO_ERROR, ?);
      inst4 : application_instance := create_entity_instance('ESTS.IOTA',
         modl, NO_ERROR, ?);
      aggr : aggregate_instance; -- LIST [1:?] OF nu
      non_persist_list_schemas : non_persistent_list_instance :=
                                 create_non_persistent_list(NO_ERROR, ?);
      non_persist_list_instances : non_persistent_list_instance :=
                                   create_non_persistent_list(NO_ERROR, ?);
      count : INTEGER;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_find_instance_roles');

   (* Instance inst2 references instance inst1. *)
   put_attribute(inst2, 'ESTS.EPSILON.E1', asp(inst1, ?), NO_ERROR, ?);

   (* Instance inst3 references instance inst1 (through the aggregate and
      through the attribute of entity type). *)
   aggr := create_aggregate_instance(inst3, 'ESTS.EPSILON.E2', ?,
      NO_ERROR, ?);
   add_by_index(aggr, 1, asp(inst1, ?), NO_ERROR, ?);
   add_by_index(aggr, 2, asp(inst1, ?), NO_ERROR, ?);
   put_attribute(inst3, 'ESTS.EPSILON.E5', asp(inst1, ?), NO_ERROR, ?);

   purpose('Ensures that find_instance_roles reports an AI_NEXS error '
      + 'when a non-persistent list for writing the result '
      + 'is not provided.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   find_instance_roles(inst1, non_persist_list_schemas, ?, AI_NEXS, ?);
   verdict;

   purpose('Ensures that find_instance_roles reports an AI_NVLD error '
     + 'when an aggregate different than non-persistent list for writing '
     + 'the result is submitted.');
   find_instance_roles(inst1, non_persist_list_schemas, aggr,
     AI_NVLD, aggr);
   verdict;

   purpose('Ensures that find_instance_roles reports an AI_NEXS error '
     + 'when a non-persistent list specifying the domain of '
     + 'entity instances is not submitted.');
   find_instance_roles(inst1, ?, non_persist_list_instances, AI_NEXS, ?);
   verdict;


   purpose('Ensures that find_instance_roles performs correctly when '
     + 'all parameters are correct.');
   find_instance_roles(inst1, non_persist_list_schemas,
     non_persist_list_instances, NO_ERROR, ?);
   verdict;

   purpose('Ensures that the resulting aggregate is empty (because a '
     + 'non-persistent list specifying the domain of entity instances '
     + 'is empty).');
   count := get_member_count(non_persist_list_instances, NO_ERROR, ?);
   assert(count = 0);
   verdict;

   (* Making the non-persistent list specifying the domain of entity
      instances to contain a given schema instance. *)
   add_by_index(non_persist_list_schemas, 1, asp(schema_inst, ?),
     NO_ERROR, ?);

   purpose('Ensures that find_instance_roles performs correctly '
     + 'and writes the attributes referencing the specified instance '
     + 'into a submitted aggregate.');
   find_instance_roles(inst1, non_persist_list_schemas,
     non_persist_list_instances, NO_ERROR, ?);
   bool := macro_compare_aggregates(non_persist_list_instances,
     ['ESTS.EPSILON.E1', 'ESTS.EPSILON.E2', 'ESTS.EPSILON.E5']);
   assert(bool);
   verdict;

   (* Instance inst1 references instance inst4. *)
   put_attribute(inst1, 'ESTS.OMEGA.O0', asp(inst4, ?), NO_ERROR, ?);

   purpose('Ensures that find_instance_roles performs correctly when a '
     + 'non-persistent list submitted for apending attributes '
     + 'is nonempty.');
   find_instance_roles(inst4, non_persist_list_schemas,
     non_persist_list_instances, NO_ERROR, ?);
   verdict;

   purpose('Ensures that the new attribute is added to a '
     + 'non-persistent list submitted for writing the result and '
     + 'all attributes earlier included into it are retained.');
   bool := is_member(non_persist_list_instances, asp('ESTS.OMEGA.O0', ?),
     NO_ERROR, ?);
   assert(bool);
   count := get_member_count(non_persist_list_instances, NO_ERROR, ?);
   assert(count = 4);
   verdict;

END_PROCEDURE; -- atc_find_instance_roles
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

(*

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema and associated with a schema instance whose native schema is ESTS schema.

6.2.54 atc_copy_application_instance

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *copy application instance*. An entity instance is copied into a model created within the same repository.

EXPRESS specification:

```
*)
PROCEDURE atc_copy_application_instance(modl : sdai_model);
  LOCAL
    model_target : sdai_model := create_sdai_model(modl.repository,
      'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
    inst1 : application_instance := create_entity_instance(
      'ESTS.EPSILON', modl, NO_ERROR, ?);
    inst2 : application_instance := create_entity_instance('ESTS.OMEGA',
      model_target, NO_ERROR, ?);
    inst3 : application_instance;
    aggr1 : aggregate_instance; -- LIST [1:?] OF nu
    aggr2 : aggregate_instance; -- LIST [1:?] OF nu
    aggr1_element1 : aggregate_primitive; -- LIST [1:3] OF REAL
    aggr2_element1 : aggregate_primitive; -- LIST [1:3] OF REAL
    aggr1_element2 : primitive;
    aggr2_element2 : primitive;
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_copy_application_instance');

  -- preparation of the test data
  (* An instance to be copied is prepared. An attribute of entity type is
     set with value. *)
  put_attribute(inst1, 'ESTS.EPSILON.E1', asp(inst2, ?), NO_ERROR, ?);

  (* An attribute of aggregation type is set with value that is an
     aggregate containing two members one of which is again aggregate. *)
  aggr1 := create_aggregate_instance(inst1, 'ESTS.EPSILON.E2', ?,
    NO_ERROR, ?);
  aggr1_element1 := add_aggregate_instance_by_index(aggr1, 1,
    ['ESTS.CHI'], NO_ERROR, ?);
  add_by_index(aggr1, 2, asp(tau.stigma, ['ESTS.TAU']), NO_ERROR, ?);

  (* An attribute of simple type is set with value. *)
  put_attribute(inst1, 'ESTS.EPSILON.E3', asp(7, ['ESTS.XI']),
    NO_ERROR, ?);

  -- starting of tests
  purpose('Ensures that copy_application_instance reports an MX_NRW '
    + 'error when a target model to which the specified instance needs '
    + 'to be copied is not in read-write mode.');
  inst3 := copy_application_instance(inst1, model_target, MX_NRW,
    model_target);
  verdict;

  (* Read-write mode for the target model is started. *)
  start_read_write_access(model_target, NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that copy_application_instance works correctly '
     + 'when parameters are correct.');
   inst3 := copy_application_instance(inst1, model_target, NO_ERROR, ?);
   verdict;

   purpose('Ensures that both entity instances, original and its '
     + 'copy, reference the same instance.');
   p := get_attribute(inst3, 'ESTS.EPSILON.E1', NO_ERROR, ?);
   assert(p :=: asp(inst2, ?));
   verdict;

   purpose('Ensures that original entity instance and its copy '
     + 'have different aggregates as values of the same attribute.');
   add_by_index(aggr1, 3, asp(100, ['ESTS.XI']), NO_ERROR, ?);
   p := get_attribute(inst3, 'ESTS.EPSILON.E2', NO_ERROR, ?);
   aggr2 := macro_convert_primitive_to_aggregate(p);
   bool := is_member(aggr2, asp(100, ['ESTS.XI']), NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that original entity instance and its copy '
     + 'have different aggregates at any level of nesting.');
   add_by_index(aggr1_element1, 1, asp(7.7, ?), NO_ERROR, ?);
   p := get_by_index(aggr2, 1, NO_ERROR, ?);
   aggr2_element1 := macro_convert_primitive_to_aggregate(p);
   bool := is_member(aggr2_element1, asp(7.7, ?), NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that original entity instance and its copy have '
     + 'the same values of simple types at any level of nesting.');
   aggr1_element2 := get_by_index(aggr1, 2, NO_ERROR, ?);
   aggr2_element2 := get_by_index(aggr2, 2, NO_ERROR, ?);
   assert(aggr1_element2 :=: aggr2_element2);
   verdict;

   purpose('Ensures that original entity instance and its copy have '
     + 'the same values of simple types.');
   p := get_attribute(inst3, 'ESTS.EPSILON.E3', NO_ERROR, ?);
   assert(p = asp(7, ?));
   verdict;

END_PROCEDURE; -- atc_copy_application_instance
(*
```

<u>Argument definitions:</u>

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema and associated with a schema instance whose native schema is ESTS schema.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.55 atc_delete_application_instance

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *delete application instance*.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_delete_application_instance(modl : sdai_model);
  LOCAL
    model_created : sdai_model := create_sdai_model(modl.repository,
                     'exceptional_name_within_repo', 'ESTS', NO_ERROR, ?);
```

```
      extent : entity_extent;
      def : entity_definition;
      inst1 : application_instance := create_entity_instance(
        'ESTS.EPSILON', modl, NO_ERROR, ?);
      inst2 : application_instance := create_entity_instance('ESTS.OMEGA',
        model_created, NO_ERROR, ?);
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_delete_application_instance');

   (* An instance references another instance that is planned
      to be deleted. *)
   put_attribute(inst1, 'ESTS.EPSILON.E1', asp(inst2, ?), NO_ERROR, ?);

   (* Saving the entity definition of the instance to be deleted. *)
   def := get_instance_type(inst2, NO_ERROR, ?);

   purpose('Deleting of an application instance.');
   delete_application_instance(inst2, NO_ERROR, ?);
   verdict;

   purpose('Ensures that a reference to an application instance that '
      + 'was deleted becomes unset.');
   bool := test_attribute(inst1, 'ESTS.EPSILON.E1', NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that deleted application instance does not belong to '
      + 'instances set of the sdai_model_contents.');
   bool := inst2 IN model_created.contents.instances;
   assert(NOT bool);
   verdict;

   purpose('Ensures that emptied entity extent does not belong to '
      + 'populated_folders set of the sdai_model_contents.');
   extent := macro_get_entity_extent(def);
   bool := macro_check_extent_if_populated(extent, model_created);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_delete_application_instance
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema and associated with a schema instance whose native schema is ESTS schema.

6.2.56 atc_validate_required_explicit_attributes_assigned

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *validate required explicit attributes assigned.*

EXPRESS specification:

```
*)
PROCEDURE atc_validate_required_explicit_attributes_assigned
        (modl : sdai_model);
   LOCAL
      inst1 : application_instance := create_entity_instance('ESTS.BETA',
        modl, NO_ERROR, ?);
      inst2 : application_instance := create_entity_instance('ESTS.KAPPA',
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      modl, NO_ERROR, ?);
    inst3 : application_instance := create_entity_instance('ESTS.ZETA',
      modl, NO_ERROR, ?);
    inst4 : application_instance := create_entity_instance(
      'ESTS.EPSILON', modl, NO_ERROR, ?);
    aggr : aggregate_instance; -- LIST [1:?] OF nu
    non_persist_list_attributes : non_persistent_list_instance :=
                                  create_non_persistent_list(NO_ERROR, ?);
    bool : boolean;
  END_LOCAL;

  atc('atc_validate_required_explicit_attributes_assigned');

  (* An instance to be validated is prepared. Two its attributes are
     set with values. *)
  put_attribute(inst1, 'ESTS.ALPHA.A1', asp(inst2, ?), NO_ERROR, ?);
  put_attribute(inst1, 'ESTS.ALPHA.A2', asp(inst3, ?), NO_ERROR, ?);

  purpose('Ensures that validate_required_explicit_attributes_assigned '
    + 'reports an AI_NEXS error when a non-persistent list for writing '
    + 'the result is not provided.');
  validate_required_explicit_attributes_assigned(inst1, ?, AI_NEXS, ?,
    bool);
  verdict;

  purpose('Ensures that validate_required_explicit_attributes_assigned '
    + 'reports an AI_NVLD error when an aggregate different than '
    + 'non-persistent list for writing the result is submitted.');
  aggr := create_aggregate_instance(inst4, 'ESTS.EPSILON.E2', ?,
    NO_ERROR, ?);
  validate_required_explicit_attributes_assigned(inst1, aggr, AI_NVLD,
    aggr, bool);
  verdict;

  purpose('Ensures that validate_required_explicit_attributes_assigned '
    + 'performs correctly when return value is FALSE.');
  validate_required_explicit_attributes_assigned(inst1,
    non_persist_list_attributes, NO_ERROR, ?, bool);
  assert(NOT bool);
  bool := macro_compare_aggregates(non_persist_list_attributes,
    ['ESTS.BETA.XXX', 'ESTS.BETA.YYY']);
  assert(bool);
  verdict;

  purpose('Ensures that attributes are added to a non-persistent list '
    + 'submitted for writing the result and all attributes '
    + 'earlier included into it are retained.');
  validate_required_explicit_attributes_assigned(inst1,
    non_persist_list_attributes, NO_ERROR, ?, bool);
  bool := macro_compare_aggregates(non_persist_list_attributes,
    ['ESTS.BETA.XXX', 'ESTS.BETA.YYY', 'ESTS.BETA.XXX',
    'ESTS.BETA.YYY']);
  assert(bool);
  verdict;

  (* Other attributes of the instance under validation are set with
     values. *)
  put_attribute(inst1, 'ESTS.BETA.XXX', asp(100, ?), NO_ERROR, ?);
  put_attribute(inst1, 'ESTS.BETA.YYY', asp(99.9, ?), NO_ERROR, ?);

  purpose('Ensures that validate_required_explicit_attributes_assigned '
    + 'performs correctly when return value is TRUE.');
  macro_clear_aggregate(non_persist_list_attributes);
  validate_required_explicit_attributes_assigned(inst1,
    non_persist_list_attributes, NO_ERROR, ?, bool);
  assert(bool);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   assert(get_member_count(non_persist_list_attributes, NO_ERROR, ?) = 0);
   verdict;

END_PROCEDURE; -- atc_validate_required_explicit_attributes_assigned
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.57 atc_validate_inverse_attributes

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *validate inverse attributes*.

EXPRESS specification:

```
*)
PROCEDURE atc_validate_inverse_attributes(modl : sdai_model);
   LOCAL
      inst1 : application_instance := create_entity_instance('ESTS.THETA',
         modl, NO_ERROR, ?);
      inst2 : application_instance := create_entity_instance('ESTS.ALPHA',
         modl, NO_ERROR, ?);
      inst3 : application_instance;
      inst4 : application_instance := create_entity_instance(
         'ESTS.EPSILON', modl, NO_ERROR, ?);
      aggr : aggregate_instance; -- LIST [1:?] OF nu
      non_persist_list_attributes : non_persistent_list_instance :=
                                    create_non_persistent_list(NO_ERROR, ?);
      bool : boolean;
   END_LOCAL;

   atc('atc_validate_inverse_attributes');

   (* A reference to an instance to be validated is established. *)
   put_attribute(inst2, 'ESTS.ALPHA.A2', asp(inst1, ?), NO_ERROR, ?);

   purpose('Ensures that validate_inverse_attributes reports an '
      + 'AI_NEXS error when a non-persistent list for writing the result '
      + 'is not provided.');
   validate_inverse_attributes(inst1, ?, AI_NEXS, ?, bool);
   verdict;

   purpose('Ensures that validate_inverse_attributes reports an AI_NVLD '
      + 'error when an aggregate different than non-persistent list '
      + 'for writing the result is submitted.');
   aggr := create_aggregate_instance(inst4, 'ESTS.EPSILON.E2', ?,
      NO_ERROR, ?);
   validate_inverse_attributes(inst1, aggr, AI_NVLD, aggr, bool);
   verdict;

   purpose('Ensures that validate_inverse_attributes performs correctly '
      + 'when return value is FALSE.');
   validate_inverse_attributes(inst1, non_persist_list_attributes,
      NO_ERROR, ?, bool);
   assert(NOT bool);
   bool := macro_compare_aggregates(non_persist_list_attributes,
      ['ESTS.THETA.Z2']);
   assert(bool);
   verdict;

   (* Another reference to an instance under validation is established. *)
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   inst3 := create_entity_instance('ESTS.ALPHA', modl, NO_ERROR, ?);
   put_attribute(inst3, 'ESTS.ALPHA.A2', asp(inst1, ?), NO_ERROR, ?);

   purpose('Ensures that validate_inverse_attributes performs correctly '
      + 'when return value is TRUE.');
   macro_clear_aggregate(non_persist_list_attributes);
   validate_inverse_attributes(inst1, non_persist_list_attributes,
      NO_ERROR, ?, bool);
   assert(bool);
   assert(get_member_count(non_persist_list_attributes, NO_ERROR, ?) = 0);
   verdict;

END_PROCEDURE; -- atc_validate_inverse_attributes
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.58 atc_validate_explicit_attributes_references

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *validate explicit attributes references.*

EXPRESS specification:

```
*)
PROCEDURE atc_validate_explicit_attributes_references(modl : sdai_model);
   LOCAL
      inst1 : application_instance := create_entity_instance('ESTS.BETA',
         modl, NO_ERROR, ?);
      inst2 : application_instance := create_entity_instance('ESTS.KAPPA',
         modl, NO_ERROR, ?);
      inst3 : application_instance := create_entity_instance('ESTS.ZETA',
         modl, NO_ERROR, ?);
      inst4 : application_instance := create_entity_instance(
         'ESTS.EPSILON', modl, NO_ERROR, ?);
      aggr : aggregate_instance; -- LIST [1:?] OF nu
      non_persist_list_attributes : non_persistent_list_instance :=
                                    create_non_persistent_list(NO_ERROR, ?);
      logic : LOGICAL;
   END_LOCAL;

   atc('atc_validate_explicit_attributes_references');

   (* An instance to be validated is prepared. An attribute of it is
    set with value. *)
   put_attribute(inst1, 'ESTS.ALPHA.A1', asp(inst2, ?), NO_ERROR, ?);

   purpose('Ensures that validate_explicit_attributes_references reports '
      + 'an AI_NEXS error when a non-persistent list for writing the '
      + 'result is not provided.');
   validate_explicit_attributes_references(inst1, ?, AI_NEXS, ?, logic);
   verdict;

   purpose('Ensures that validate_explicit_attributes_references reports '
      + 'an AI_NVLD error when an aggregate different than non-persistent '
      + 'list for writing the result is submitted.');
   aggr := create_aggregate_instance(inst4, 'ESTS.EPSILON.E2', ?,
      NO_ERROR, ?);
   validate_explicit_attributes_references(inst1, aggr, AI_NVLD, aggr,
      logic);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
  purpose('Ensures that validate_explicit_attributes_references performs '
    + 'correctly when return value is UNKNOWN.');
  validate_explicit_attributes_references(inst1,
    non_persist_list_attributes, NO_ERROR, ?, logic);
  assert(logic = UNKNOWN);
  assert(get_member_count(non_persist_list_attributes, NO_ERROR, ?) = 0);
  verdict;

  (* Another attribute of the instance under validation is set with value.
  *)
  put_attribute(inst1, 'ESTS.ALPHA.A2', asp(inst3, ?), NO_ERROR, ?);

  purpose('Ensures that validate_explicit_attributes_references performs '
    + 'correctly when return value is TRUE.');
  validate_explicit_attributes_references(inst1,
    non_persist_list_attributes, NO_ERROR, ?, logic);
  assert(logic = TRUE);
  assert(get_member_count(non_persist_list_attributes, NO_ERROR, ?) = 0);
  verdict;

END_PROCEDURE; -- atc_validate_explicit_attributes_references
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.59 atc_validate_aggregates_size

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *validate aggregates size.*

EXPRESS specification:

```
*)
PROCEDURE atc_validate_aggregates_size(modl : sdai_model);
  LOCAL
    inst : application_instance := create_entity_instance('ESTS.EPSILON',
      modl, NO_ERROR, ?);
    aggr1 : aggregate_instance := create_aggregate_instance(inst,
            'ESTS.EPSILON.E2', ?, NO_ERROR, ?); -- LIST [1:?] OF nu
    aggr2 : aggregate_instance := create_aggregate_instance(inst,
            'ESTS.EPSILON.E6', ?, NO_ERROR, ?); -- SET [1:?] OF rho
    non_persist_list_attributes : non_persistent_list_instance :=
                                  create_non_persistent_list(NO_ERROR, ?);
    logic : LOGICAL;
    bool : boolean;
  END_LOCAL;

  atc('atc_validate_aggregates_size');

  purpose('Ensures that validate_aggregates_size reports an AI_NEXS '
    + 'error when a non-persistent list for writing the result is '
    + 'not provided.');
  validate_aggregates_size(inst, ?, AI_NEXS, ?, logic);
  verdict;

  purpose('Ensures that validate_aggregates_size reports an AI_NVLD '
    + 'error when an aggregate different than non-persistent list '
    + 'for writing the result is submitted.');
  validate_aggregates_size(inst, aggr1, AI_NVLD, aggr1, logic);
  verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
  purpose('Ensures that validate_aggregates_size performs correctly '
    + 'when return value is FALSE.');
  validate_aggregates_size(inst, non_persist_list_attributes, NO_ERROR, ?,
    logic);
  assert(logic = FALSE);
  bool := macro_compare_aggregates(non_persist_list_attributes,
    ['ESTS.EPSILON.E2', 'ESTS.EPSILON.E6']);
  assert(bool);
  verdict;

  (* An instance under validation is modified. *)
  add_by_index(aggr1, 1, asp(tau.stigma, ['ESTS.TAU']), NO_ERROR, ?);
  add_unordered(aggr2, asp(10, ['ESTS.XI']), NO_ERROR, ?);

  purpose('Ensures that validate_aggregates_size performs correctly '
    + 'when return value is TRUE.');
  macro_clear_aggregate(non_persist_list_attributes);
  validate_aggregates_size(inst, non_persist_list_attributes, NO_ERROR, ?,
    logic);
  assert(logic = TRUE);
  assert(get_member_count(non_persist_list_attributes, NO_ERROR, ?) = 0);
  verdict;

END_PROCEDURE; -- atc_validate_aggregates_size
(*
```

<u>Argument definitions:</u>

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.60 atc_validate_aggregates_uniqueness

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *validate aggregates uniqueness*.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_validate_aggregates_uniqueness(modl : sdai_model);
  LOCAL
    inst : application_instance := create_entity_instance('ESTS.EPSILON',
      modl, NO_ERROR, ?);
    aggr : aggregate_instance := create_aggregate_instance(inst,
           'ESTS.EPSILON.E6', ?, NO_ERROR, ?); -- SET [1:?] OF rho
    non_persist_list_attributes : non_persistent_list_instance :=
                                  create_non_persistent_list(NO_ERROR, ?);
    bool : boolean;
  END_LOCAL;

  atc('atc_validate_aggregates_uniqueness');

  (* An instance to be validated is prepared. *)
  add_unordered(aggr, asp(10, ['ESTS.XI']), NO_ERROR, ?);
  add_unordered(aggr, asp(20, ['ESTS.XI']), NO_ERROR, ?);

  purpose('Ensures that validate_aggregates_uniqueness reports an '
    + 'AI_NEXS error when a non-persistent list for writing the result '
    + 'is not provided.');
  validate_aggregates_uniqueness(inst, ?, AI_NEXS, ?, bool);
  verdict;

  purpose('Ensures that validate_aggregates_uniqueness reports an '
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      + 'AI_NVLD error when an aggregate different than non-persistent '
      + 'list for writing the result is submitted.');
   validate_aggregates_uniqueness(inst, aggr, AI_NVLD, aggr, bool);
   verdict;

   purpose('Ensures that validate_aggregates_uniqueness performs '
      + 'correctly when return value is TRUE.');
   validate_aggregates_uniqueness(inst, non_persist_list_attributes,
      NO_ERROR, ?, bool);
   assert(bool);
   assert(get_member_count(non_persist_list_attributes, NO_ERROR, ?) = 0);
   verdict;

   (* An instance under validation is modified. *)
   add_unordered(aggr, asp(10, ['ESTS.XI']), NO_ERROR, ?);

   purpose('Ensures that validate_aggregates_uniqueness performs '
      + 'correctly when return value is FALSE.');
   validate_aggregates_uniqueness(inst, non_persist_list_attributes,
      NO_ERROR, ?, bool);
   assert(NOT bool);
   bool := macro_compare_aggregates(non_persist_list_attributes,
      ['ESTS.EPSILON.E6']);
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_validate_aggregates_uniqueness
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.61 atc_validate_array_not_optional

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *validate array not optional.*

EXPRESS specification:

```
*)
PROCEDURE atc_validate_array_not_optional(modl : sdai_model);
   LOCAL
      inst : application_instance := create_entity_instance('ESTS.EPSILON',
         modl, NO_ERROR, ?);
      aggr : aggregate_instance := create_aggregate_instance(inst,
            'ESTS.EPSILON.E4', ?, NO_ERROR, ?); -- ARRAY [1:3] OF omicron
      non_persist_list_attributes : non_persistent_list_instance :=
                                    create_non_persistent_list(NO_ERROR, ?);
      bool : boolean;
   END_LOCAL;

   atc('atc_validate_array_not_optional');

   (* An instance to be validated is prepared. *)
   put_by_index(aggr, 1, asp(tau.stigma, ['ESTS.TAU']), NO_ERROR, ?);
   put_by_index(aggr, 3, asp(30, ['ESTS.XI']), NO_ERROR, ?);

   purpose('Ensures that validate_array_not_optional reports an '
      + 'AI_NEXS error when a non-persistent list for writing the result '
      + 'is not provided.');
   validate_array_not_optional(inst, ?, AI_NEXS, ?, bool);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   purpose('Ensures that validate_array_not_optional reports an '
      + 'AI_NVLD error when an aggregate different than non-persistent '
      + 'list for writing the result is submitted.');
   validate_array_not_optional(inst, aggr, AI_NVLD, aggr, bool);
   verdict;

   purpose('Ensures that validate_array_not_optional performs '
      + 'correctly when return value is FALSE.');
   validate_array_not_optional(inst, non_persist_list_attributes,
      NO_ERROR, ?, bool);
   assert(NOT bool);
   bool := macro_compare_aggregates(non_persist_list_attributes,
      ['ESTS.EPSILON.E4']);
   assert(bool);
   verdict;

   (* An instance under validation is modified. *)
   put_by_index(aggr, 2, asp(20, ['ESTS.XI']), NO_ERROR, ?);

   purpose('Ensures that validate_array_not_optional performs '
      + 'correctly when return value is TRUE.');
   macro_clear_aggregate(non_persist_list_attributes);
   validate_array_not_optional(inst, non_persist_list_attributes,
      NO_ERROR, ?, bool);
   assert(bool);
   assert(get_member_count(non_persist_list_attributes, NO_ERROR, ?) = 0);
   verdict;

END_PROCEDURE; -- atc_validate_array_not_optional
(*
```

<u>Argument definitions:</u>

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.62 atg_aggregate_simple

This abstract test group shall be used for the assesment of the implementation of the SDAI operations for non-nested aggregates.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atg_aggregate_simple(modl : sdai_model);
   LOCAL
      inst1 : application_instance := create_entity_instance('ESTS.SIGMA',
         modl, NO_ERROR, ?);
      inst2 : application_instance := create_entity_instance(
         'ESTS.EPSILON', modl, NO_ERROR, ?);
      aggr : aggregate_instance;
      iter : iterator;
   END_LOCAL;

   (* An aggregate of type LIST of NUMBER and its iterator are created. *)
   aggr := create_aggregate_instance(inst1, 'ESTS.SIGMA.S1', ?,
      NO_ERROR, ?);
   iter := create_iterator(aggr, NO_ERROR, ?);

   (* Testing of the SDAI operations on iterator creation, deletion and
      positioning. *)
   atg_iterator(aggr, iter);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
(* Checks cases when aggregate operations are illegal for the LIST. *)
atc_aggregate_operations_disallowed_for_list(aggr, iter);

(* Testing the basic functionality of the SDAI operations for an
   aggregate of type LIST of NUMBER. *)
atg_list_number(aggr, iter);

(* An aggregate of type LIST of entity instances and its iterator
   are created. *)
aggr := create_aggregate_instance(inst2, 'ESTS.EPSILON.E2', ?,
  NO_ERROR, ?);
iter := create_iterator(aggr, NO_ERROR, ?);

(* Testing the basic functionality of the SDAI operations for
 an aggregate of type LIST of entity instances. *)
atg_list_entity(aggr, iter, modl);

(* An aggregate of type SET of LOGICAL and its iterator are created. *)
aggr := create_aggregate_instance(inst1, 'ESTS.SIGMA.S4', ?,
  NO_ERROR, ?);
iter := create_iterator(aggr, NO_ERROR, ?);

(* Checks cases when aggregate operations are illegal for the SET. *)
atc_aggregate_operations_disallowed_for_set(aggr, iter);

(* Testing the basic functionality of the SDAI operations for
   an aggregate of type SET of LOGICAL. *)
atg_set_logical(aggr, iter);

(* An aggregate of type SET of EXPRESS TYPE and its iterator are
   created. *)
aggr := create_aggregate_instance(inst1, 'ESTS.SIGMA.S9', ?,
  NO_ERROR, ?);
iter := create_iterator(aggr, NO_ERROR, ?);

(* Testing the basic functionality of the SDAI operations for
   an aggregate of type SET of EXPRESS TYPE. *)
atg_set_simple_defined_type(aggr, iter);

(* An aggregate of type BAG of STRING and its iterator are created. *)
aggr := create_aggregate_instance(inst1, 'ESTS.SIGMA.S6', ?,
  NO_ERROR, ?);
iter := create_iterator(aggr, NO_ERROR, ?);

(* Testing iterator operation next for the unordered collection. *)
atc_next_iterator_for_unordered_collection(aggr, iter);

(* Testing the basic functionality of the SDAI operations for
   an aggregate of type BAG of STRING. *)
atg_bag_of_string(aggr, iter);

(* An aggregate of type BAG of REAL and its iterator are created. *)
aggr := create_aggregate_instance(inst1, 'ESTS.SIGMA.S2', ?,
  NO_ERROR, ?);
iter := create_iterator(aggr, NO_ERROR, ?);

(* Testing the basic functionality of the SDAI operations for
   an aggregate of type BAG of REAL. *)
atg_bag_of_real(aggr, iter);

(* An aggregate of type ARRAY of INTEGER and its iterator are created.
*)
aggr := create_aggregate_instance(inst1, 'ESTS.SIGMA.S3', ?,
  NO_ERROR, ?);
iter := create_iterator(aggr, NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   (* Checks cases when aggregate operations are illegal for the ARRAY. *)
   atc_aggregate_operations_disallowed_for_array(aggr, iter);

   (* Testing the basic functionality of the SDAI operations for
      an aggregate of type ARRAY of INTEGER. *)
   atg_array_of_integer(aggr, iter);

END_PROCEDURE; -- atg_aggregate_simple
(*
```

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.63 atg_iterator

This abstract test group shall be used for the assesment of the implementation of the SDAI operations operations *create iterator*, *delete iterator*, *beginning*, *end*, *next*, and *previous*.

EXPRESS specification:

```
*)
PROCEDURE atg_iterator(aggr : aggregate_instance; iter : iterator);

   (* Testing iterator operations create iterator and delete iterator. *)
   atc_create_iterator_delete_iterator(aggr);

   (* Testing iterator operations beginning and end. *)
   atc_beginning_end_iterator(iter);

   (* Testing iterator operation next for the ordered collection. *)
   atc_next_iterator_for_ordered_collection(iter);

   (* Testing iterator operation previous. *)
   atc_previous_iterator(iter);

END_PROCEDURE; -- atg_iterator
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of NUMBER. The aggregate shall be empty initally.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.64 atc_create_iterator_delete_iterator

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *create iterator* and *delete iterator*.

EXPRESS specification:

```
*)
PROCEDURE atc_create_iterator_delete_iterator(aggr : aggregate_instance);
   LOCAL
      iter : iterator;
      bool : BOOLEAN;
   END_LOCAL;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   atc('atc_create_iterator_delete_iterator');

   (* Making the aggregate to contain one element. *)
   add_by_index(aggr, 1, asp(5.0, ?), NO_ERROR, ?);

   purpose('Ensures that create_iterator works correctly.');
   iter := create_iterator(aggr, NO_ERROR, ?);
   verdict;

   purpose('Ensures that newly created iterator is positioned at '
      + 'the beginning and has no current member.');
   bool := next(iter, NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that delete_iterator reports an IR_NEXS error '
      + 'when iterator is not provided.');
   delete_iterator(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that delete_iterator works correctly when iterator '
      + 'is submitted.');
   delete_iterator(iter, NO_ERROR, ?);
   verdict;

   purpose('Ensures that iterator was deleted by the '
      + 'delete_iterator operation.');
   beginning(iter, IR_NEXS, ?);
   verdict;

END_PROCEDURE; -- atc_create_iterator_delete_iterator
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of NUMBER. The aggregate shall be empty.

6.2.65 atc_beginning_end_iterator

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *beginning* and *end*.

EXPRESS specification:

```
*)
PROCEDURE atc_beginning_end_iterator(iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_beginning_end_iterator');

   purpose('Ensures that beginning reports an IR_NEXS error when iterator '
      + 'is not provided.');
   beginning(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that after beginning operation iterator is '
      + 'positioned at the beginning of the aggregate (there is no '
      + 'current member).');
   beginning(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   bool := next(iter, NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that atEnd reports an IR_NEXS error when iterator '
      + 'is not provided.');
   atEnd(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that after end operation iterator is positioned at '
      + 'the end of the aggregate (there is no current member).');
   atEnd(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
   bool := previous(iter, NO_ERROR, ?);
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_beginning_end_iterator
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_create_iterator_delete_iterator.

6.2.66 atc_next_iterator_for_ordered_collection

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *next* for the case when the attached aggregate is an ordered collection.

EXPRESS specification:

```
*)
PROCEDURE atc_next_iterator_for_ordered_collection(iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_next_iterator_for_ordered_collection');

   purpose('Ensures that next reports an IR_NEXS error when iterator '
      + 'is not provided.');
   bool := next(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that next works correctly when before this operation '
      + 'iterator is at the beginning.');
   beginning(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p = asp(5.0, ?));
   verdict;

   purpose('Ensures that next works correctly when before this '
      + 'operation iterator is at the end.');
   atEnd(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   assert(NOT bool);
   bool := previous(iter, NO_ERROR, ?);
   assert(bool);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that next works correctly when before this '
      + 'operation iterator is at the last member of the aggregate.');
   bool := next(iter, NO_ERROR, ?);
   assert(NOT bool);
   bool := previous(iter, NO_ERROR, ?);
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_next_iterator_for_ordered_collection
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_create_iterator_delete_iterator.

6.2.67 atc_previous_iterator

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *previous*.

EXPRESS specification:

```
*)
PROCEDURE atc_previous_iterator(iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_previous_iterator');

   purpose('Ensures that previous reports an IR_NEXS error when iterator '
      + 'is not provided.');
   bool := previous(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that previous works correctly when before this '
      + 'operation iterator is at the end.');
   atEnd(iter, NO_ERROR, ?);
   bool := previous(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p = asp(5.0, ?));
   verdict;

   purpose('Ensures that previous works correctly when before this '
      + 'operation iterator is at the beginning.');
   beginning(iter, NO_ERROR, ?);
   bool := previous(iter, NO_ERROR, ?);
   assert(NOT bool);
   bool := next(iter, NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that previous works correctly when before this '
      + 'operation iterator is at the first member of the aggregate.');
   bool := previous(iter, NO_ERROR, ?);
   assert(NOT bool);
   bool := next(iter, NO_ERROR, ?);
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_previous_iterator
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_create_iterator_delete_iterator.

6.2.68 atc_next_iterator_for_unordered_collection

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *next* for the case when the attached aggregate is an unordered collection.

EXPRESS specification:

```
*)
PROCEDURE atc_next_iterator_for_unordered_collection
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    bool : BOOLEAN;
    p1, p2 : primitive;
  END_LOCAL;

  atc('atc_next_iterator_for_unordered_collection');

  (* Adding two values to the aggregate. *)
  add_unordered(aggr, asp('first', ?), NO_ERROR, ?);
  add_unordered(aggr, asp('second', ?), NO_ERROR, ?);

  purpose('Ensures that for an unordered collection the repeated '
    + 'application of the next operation gives all members of '
    + 'the collection.');
  beginning(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  p1 := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  p2 := get_current_member(iter, NO_ERROR, ?);
  bool := macro_compare_aggregates(aggr, [p1, p2]);
  assert(bool);
  verdict;

END_PROCEDURE; -- atc_next_iterator_for_unordered_collection
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of STRING.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.69 atg_list_number

This abstract test group shall be used for the assesment of the implementation of the SDAI operations *get by index*, *put by index*, *add by index*, *remove by index*, *is member*, *add before current member*, *add after current member*, *get current member*, *put current member*, and *remove current member* for an aggregate of type LIST of NUMBER.

EXPRESS specification:

```
*)
PROCEDURE atg_list_number(aggr : aggregate_instance; iter : iterator);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   (* Making the given aggregate empty. *)
   macro_clear_aggregate(aggr);

   (* Testing of the list operation add by index. *)
   atc_add_by_index_list_number(aggr);

   (* Testing of the aggregate operation get by index. *)
   atc_get_by_index_list_number(aggr);

   (* Testing of the aggregate operation put by index. *)
   atc_put_by_index_list_number(aggr);

   (* Testing of the aggregate operation is member. *)
   atc_is_member_list_number(aggr);

   (* Testing of the list operation remove by index. *)
   atc_remove_by_index_list_number(aggr);

   (* Making the given aggregate empty. *)
   macro_clear_aggregate(aggr);

   (* Testing of the list operation add before current member. *)
   atc_add_before_current_member_list_number(iter);

   (* Testing of the list operation add after current member. *)
   atc_add_after_current_member_list_number(iter);

   (* Testing of the aggregate operation get current member. *)
   atc_get_current_member_list_number(iter);

   (* Testing of the aggregate operation put current member. *)
   atc_put_current_member_list_number(iter);

   (* Testing of the aggregate operation remove current member. *)
   atc_remove_current_member_list_number(iter);

END_PROCEDURE; -- atg_list_number
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of NUMBER.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.70 atc_add_by_index_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add by index* for the case when the aggregate is of type LIST of NUMBER.

EXPRESS specification:

```
*)
PROCEDURE atc_add_by_index_list_number(aggr : aggregate_instance);

   atc('atc_add_by_index_list_number');

   purpose('Ensures that indexing in the LIST starts from 1.');
   add_by_index(aggr, 0, asp(1.5, ?), IX_NVLD, aggr);
   verdict;

   purpose('Ensures that add_by_index reports a VA_NSET error when an '
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      + 'unset value is submitted.');
   add_by_index(aggr, 1, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that add_by_index reports a VT_NVLD error when value '
      + 'of a wrong type is submitted.');
   add_by_index(aggr, 1, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that add_by_index works correctly when all parameters '
      + 'are correct.');
   add_by_index(aggr, 1, asp(1.5, ?), NO_ERROR, ?);
   add_by_index(aggr, 2, asp(77, ?), NO_ERROR, ?);
   add_by_index(aggr, 1, asp(9.99, ?), NO_ERROR, ?);
   add_by_index(aggr, 3, asp(3.14, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that add_by_index reports an IX_NVLD error when the '
      + 'index submitted exceeds the count of aggregate members plus one.');
   add_by_index(aggr, 6, asp(1.5, ?), IX_NVLD, aggr);
   verdict;

END_PROCEDURE; -- atc_add_by_index_list_number
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of NUMBER. The aggregate submitted shall be empty.

6.2.71 atc_get_by_index_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get by index* for the case when the aggregate is of type LIST of NUMBER.

EXPRESS specification:

```
*)
PROCEDURE atc_get_by_index_list_number(aggr : aggregate_instance);
   LOCAL
      p : primitive;
   END_LOCAL;

   atc('atc_get_by_index_list_number');

   purpose('Ensures that get_by_index reports an IX_NVLD error when the '
      + 'index submitted is outside of the legal range.');
   p := get_by_index(aggr, 0, IX_NVLD, aggr);
   p := get_by_index(aggr, 5, IX_NVLD, aggr);
   verdict;

   purpose('Ensures that get_by_index works correctly when the index '
      + 'submitted is from the legal range.');
   p := get_by_index(aggr, 1, NO_ERROR, ?);
   assert(p = asp(9.99, ?));
   p := get_by_index(aggr, 2, NO_ERROR, ?);
   assert(p = asp(1.5, ?));
   p := get_by_index(aggr, 3, NO_ERROR, ?);
   assert(p = asp(3.14, ?));
   p := get_by_index(aggr, 4, NO_ERROR, ?);
   assert(p = asp(77, ?));
   verdict;

END_PROCEDURE; -- atc_get_by_index_list_number
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_add_by_index_list_number.

6.2.72 atc_put_by_index_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *put by index* for the case when the aggregate is of type LIST of NUMBER.

EXPRESS specification:

```
*)
PROCEDURE atc_put_by_index_list_number(aggr : aggregate_instance);
   LOCAL
      p : primitive;
   END_LOCAL;

   atc('atc_put_by_index_list_number');

   purpose('Ensures that put_by_index reports an IX_NVLD error when the '
      + 'index submitted is outside of the legal range.');
   put_by_index(aggr, 0, asp(3.147, ?), IX_NVLD, aggr);
   put_by_index(aggr, 5, asp(3.147, ?), IX_NVLD, aggr);
   verdict;

   purpose('Ensures that put_by_index reports a VA_NSET error when an '
      + 'unset value is submitted.');
   put_by_index(aggr, 3, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_by_index reports a VT_NVLD error when value '
      + 'of a wrong type is submitted.');
   put_by_index(aggr, 3, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_by_index works correctly when all parameters '
       + 'are correct.');
   put_by_index(aggr, 3, asp(3.147, ?), NO_ERROR, ?);
   p := get_by_index(aggr, 3, NO_ERROR, ?);
   assert(p = asp(3.147, ?));
   verdict;

END_PROCEDURE; -- atc_put_by_index_list_number
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_add_by_index_list_number.

6.2.73 atc_is_member_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *is member* and *get member count* for the case when the aggregate is of type LIST of NUMBER.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
*)
PROCEDURE atc_is_member_list_number(aggr : aggregate_instance);
  LOCAL
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_is_member_list_number');

  purpose('Ensures that is_member returns TRUE when value submitted '
    + 'belongs to the aggregate.');
  bool := is_member(aggr, asp(77.0, ?), NO_ERROR, ?);
  assert(bool);
  verdict;

  purpose('Ensures that is_member returns FALSE when value submitted '
    + 'does not belong to the aggregate.');
  bool := is_member(aggr, asp(9.9, ?), NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that get_member_count works correctly.');
  assert(get_member_count(aggr, NO_ERROR, ?) = 4);
  verdict;

END_PROCEDURE; -- atc_is_member_list_number
(*
```

<ins>Argument definitions:</ins>

aggr: (input) An aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_put_by_index_list_number.

6.2.74 atc_remove_by_index_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove by index* for the case when the aggregate is of type LIST of NUMBER.

<ins>EXPRESS specification:</ins>

```
*)
PROCEDURE atc_remove_by_index_list_number(aggr : aggregate_instance);
  LOCAL
    p : primitive;
  END_LOCAL;

  atc('atc_remove_by_index_list_number');

  purpose('Ensures that remove_by_index reports an IX_NVLD error when '
    + 'the index submitted is outside of the legal range.');
  remove_by_index(aggr, 0, IX_NVLD, aggr);
  remove_by_index(aggr, 5, IX_NVLD, aggr);
  verdict;

  purpose('Ensures that remove_by_index works correctly when the '
    + 'index submitted is from a legal range.');
  remove_by_index(aggr, 2, NO_ERROR, ?);
  p := get_by_index(aggr, 2, NO_ERROR, ?);
  assert(p = asp(3.147, ?));
  verdict;

END_PROCEDURE; -- atc_remove_by_index_list_number
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Argument definitions:

aggr: (input) An aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_put_by_index_list_number.

6.2.75 atc_add_before_current_member_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add before current member* for an aggregate of type LIST of NUMBER and of the SDAI operation *add after current member* for an empty aggregate.

EXPRESS specification:

```
*)
PROCEDURE atc_add_before_current_member_list_number(iter : iterator);
  LOCAL
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_add_before_current_member_list_number');

  purpose('Ensures that after add_after_current_member operation for an '
    + 'empty list iterator is positioned at the beginning of the list '
    + '(there is no current member).');
  atEnd(iter, NO_ERROR, ?);
  add_after_current_member(iter, asp(59.5, ?), NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  assert(bool);
  bool := remove_current_member(iter, NO_ERROR, ?);
  verdict;

  purpose('Ensures that add_before_current_member reports an IR_NEXS '
    + 'error when iterator is not provided.');
  add_before_current_member(?, asp(3.3, ?), IR_NEXS, ?);
  verdict;

  purpose('Ensures that add_before_current_member reports a VA_NSET '
    + 'error when an unset value is submitted.');
  add_before_current_member(iter, ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that add_before_current_member reports a VT_NVLD '
    + 'error when value of a wrong type is submitted.');
  add_before_current_member(iter, asp('something', ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that after add_before_current_member operation for '
    + 'an empty list iterator is positioned at the end of the list '
    + '(there is no current member).');
  beginning(iter, NO_ERROR, ?);
  add_before_current_member(iter, asp(3.3, ?), NO_ERROR, ?);
  bool := previous(iter, NO_ERROR, ?);
  assert(bool);
  verdict;

  purpose('Ensures that add_before_current_member works correctly when '
    + 'all parameters are correct.');
  add_before_current_member(iter, asp(2.2, ?), NO_ERROR, ?);
  atEnd(iter, NO_ERROR, ?);
  add_before_current_member(iter, asp(4.4, ?), NO_ERROR, ?);
  verdict;

END_PROCEDURE; -- atc_add_before_current_member_list_number
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of NUMBER. The aggregate shall be empty.

6.2.76 atc_add_after_current_member_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add after current member* for an aggregate of type LIST of NUMBER.

EXPRESS specification:

```
*)
PROCEDURE atc_add_after_current_member_list_number(iter : iterator);

  atc('atc_add_after_current_member_list_number');

  (* Positioning of the iterator at the end of the aggregate. *)
  atEnd(iter, NO_ERROR, ?);

  purpose('Ensures that add_after_current_member reports an IR_NEXS '
    + 'error when iterator is not provided.');
  add_after_current_member(?, asp(5.5, ?), IR_NEXS, ?);
  verdict;

  purpose('Ensures that add_after_current_member reports a VA_NSET '
    + 'error when an unset value is submitted.');
  add_after_current_member(iter, ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that add_after_current_member reports a VT_NVLD '
    + 'error when value of a wrong type is submitted.');
  add_after_current_member(iter, asp('something', ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that add_after_current_member works correctly when '
    + 'all parameters are correct.');
  add_after_current_member(iter, asp(5.5, ?), NO_ERROR, ?);
  add_after_current_member(iter, asp(6.6, ?), NO_ERROR, ?);
  beginning(iter, NO_ERROR, ?);
  add_after_current_member(iter, asp(1.1, ?), NO_ERROR, ?);
  verdict;

END_PROCEDURE; -- atc_add_after_current_member_list_number
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_add_before_current_member_list_number.

6.2.77 atc_get_current_member_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get current member* for an aggregate of type LIST of NUMBER.

EXPRESS specification:

```
*)
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
PROCEDURE atc_get_current_member_list_number(iter : iterator);
  LOCAL
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_get_current_member_list_number');

  purpose('Ensures that get_current_member reports an IR_NEXS error when '
     + 'iterator is not provided.');
  p := get_current_member(?, IR_NEXS, ?);
  verdict;

  purpose('Ensures that get_current_member reports an IR_NSET error when '
     + 'iterator has no current member set.');
  beginning(iter, NO_ERROR, ?);
  p := get_current_member(iter, IR_NSET, iter);
  atEnd(iter, NO_ERROR, ?);
  p := get_current_member(iter, IR_NSET, iter);
  verdict;

  purpose('Ensures that get_current_member works correctly when '
    + 'iterator has current member set.');
  bool := previous(iter, NO_ERROR, ?);
  p := get_current_member(iter, NO_ERROR, ?);
  assert(p = asp(6.6, ?));
  bool := previous(iter, NO_ERROR, ?);
  p := get_current_member(iter, NO_ERROR, ?);
  assert(p = asp(5.5, ?));
  bool := previous(iter, NO_ERROR, ?);
  p := get_current_member(iter, NO_ERROR, ?);
  assert(p = asp(4.4, ?));
  bool := previous(iter, NO_ERROR, ?);
  p := get_current_member(iter, NO_ERROR, ?);
  assert(p = asp(3.3, ?));
  bool := previous(iter, NO_ERROR, ?);
  p := get_current_member(iter, NO_ERROR, ?);
  assert(p = asp(2.2, ?));
  bool := previous(iter, NO_ERROR, ?);
  p := get_current_member(iter, NO_ERROR, ?);
  assert(p = asp(1.1, ?));
  verdict;

END_PROCEDURE; -- atc_get_current_member_list_number
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_add_after_current_member_list_number.

6.2.78 atc_put_current_member_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *put current member* for an aggregate of type LIST of NUMBER.

EXPRESS specification:

```
*)
PROCEDURE atc_put_current_member_list_number(iter : iterator);
  LOCAL
    bool : BOOLEAN;
    p : primitive;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   END_LOCAL;

   atc('atc_put_current_member_list_number');

   purpose('Ensures that put_current_member reports an IR_NEXS error when '
      + 'iterator is not provided.');
   put_current_member(?, asp(9.99, ?), IR_NEXS, ?);
   verdict;

   purpose('Ensures that put_current_member reports an IR_NSET error when '
      + 'iterator has no current member set.');
   beginning(iter, NO_ERROR, ?);
   put_current_member(iter, asp(9.99, ?), IR_NSET, iter);
   atEnd(iter, NO_ERROR, ?);
   put_current_member(iter, asp(9.99, ?), IR_NSET, iter);
   verdict;

   purpose('Ensures that put_current_member reports a VA_NSET error when '
      + 'an unset value is submitted.');
   put_current_member(iter, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_current_member reports a VT_NVLD error when '
      + 'value of a wrong type is submitted.');
   put_current_member(iter, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_current_member works correctly when '
      + 'all parameters are correct.');
   bool := previous(iter, NO_ERROR, ?);
   put_current_member(iter, asp(9.99, ?), NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p = asp(9.99, ?));
   verdict;

END_PROCEDURE; -- atc_put_current_member_list_number
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_add_after_current_member_list_number.

6.2.79 atc_remove_current_member_list_number

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove current member* for an aggregate of type LIST of NUMBER.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_current_member_list_number(iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_remove_current_member_list_number');

   purpose('Ensures that remove_current_member reports an IR_NEXS '
      + 'error when iterator is not provided.');
   bool := remove_current_member(?, IR_NEXS, ?);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that remove_current_member reports an IR_NSET '
     + 'error when iterator has no current member set.');
   atEnd(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, IR_NSET, iter);
   beginning(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, IR_NSET, iter);
   verdict;

   purpose('Ensures that remove_current_member works correctly '
     + 'when iterator has current member set.');
   bool := next(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, NO_ERROR, ?);
   assert(bool);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p = asp(2.2, ?));
   verdict;

   purpose('Ensures that remove_current_member returns FALSE when '
     + 'iterator refers to the last member of the aggregate.');
   atEnd(iter, NO_ERROR, ?);
   bool := previous(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_remove_current_member_list_number
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of NUMBER. The aggregate shall be that processed by atc_put_current_member_list_number.

6.2.80 atg_list_entity

This abstract test group shall be used for the assessment of the implementation of the SDAI operations *get by index*, *put by index*, *add by index*, *remove by index*, *is member*, *add before current member*, *add after current member*, *get current member*, *put current member*, and *remove current member* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atg_list_entity(aggr : aggregate_instance; iter : iterator;
           modl : sdai_model);
   LOCAL
     inst1 : application_instance := create_entity_instance('ESTS.OMEGA',
       modl, NO_ERROR, ?);
     inst2 : application_instance := create_entity_instance('ESTS.OMEGA',
       modl, NO_ERROR, ?);
     inst3 : application_instance := create_entity_instance('ESTS.OMEGA',
       modl, NO_ERROR, ?);
     inst4 : application_instance := create_entity_instance('ESTS.OMEGA',
       modl, NO_ERROR, ?);
     inst5 : application_instance := create_entity_instance('ESTS.OMEGA',
       modl, NO_ERROR, ?);
     inst6 : application_instance := create_entity_instance('ESTS.OMEGA',
       modl, NO_ERROR, ?);
   END_LOCAL;

   atc('atg_list_entity');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   (* Testing of the list operation add by index. *)
   atc_add_by_index_list_entity(aggr, [inst1, inst2, inst3, inst4]);

   (* Testing of the aggregate operation get by index. *)
   atc_get_by_index_list_entity(aggr, [inst1, inst2, inst3, inst4]);

   (* Testing of the aggregate operation put by index. *)
   atc_put_by_index_list_entity(aggr, inst4);

   (* Testing of the aggregate operation is member. *)
   atc_is_member_list_entity(aggr, [inst1, inst2, inst3, inst4]);

   (* Testing of the list operation remove by index. *)
   atc_remove_by_index_list_entity(aggr, inst4);

   (* Making the given aggregate empty. *)
   macro_clear_aggregate(aggr);

   (* Testing of the list operation add before current member. *)
   atc_add_before_current_member_list_entity
     (iter, [inst1, inst2, inst3, inst4]);

   (* Testing of the list operation add after current member. *)
   atc_add_after_current_member_list_entity
     (iter, [inst1, inst2, inst3, inst4, inst5, inst6]);

   (* Testing of the aggregate operation get current member. *)
   atc_get_current_member_list_entity
     (iter, [inst1, inst2, inst3, inst4, inst5, inst6]);

   (* Testing of the aggregate operation put current member. *)
   atc_put_current_member_list_entity(iter, inst1);

   (* Testing of the aggregate operation remove current member. *)
   atc_remove_current_member_list_entity(iter, inst2);

END_PROCEDURE; -- atg_list_entity
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of nu in ESTS schema. The aggregate shall be empty.

iter: (input) An iterator over an aggregate specified by the first argument.

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.81 atc_add_by_index_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add by index* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atc_add_by_index_list_entity(aggr : aggregate_instance;
     aggr_with_data : BAG [0:?] OF application_instance);

   atc('atc_add_by_index_list_entity');

   purpose('Ensures that indexing in the LIST starts from one.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   add_by_index(aggr, 0, asp(aggr_with_data[1], ?), IX_NVLD, aggr);
   verdict;

   purpose('Ensures that add_by_index reports a VA_NSET error when an '
      + 'unset value is submitted.');
   add_by_index(aggr, 1, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that add_by_index reports a VT_NVLD error when value '
      + 'of a wrong type is submitted.');
   add_by_index(aggr, 1, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that add_by_index works correctly when all parameters '
      + 'are correct.');
   add_by_index(aggr, 1, asp(aggr_with_data[2], ?), NO_ERROR, ?);
   add_by_index(aggr, 2, asp(aggr_with_data[4], ?), NO_ERROR, ?);
   add_by_index(aggr, 1, asp(aggr_with_data[1], ?), NO_ERROR, ?);
   add_by_index(aggr, 3, asp(aggr_with_data[3], ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that add_by_index reports an IX_NVLD error when '
      + 'the index submitted exceeds the count of aggregate members '
      + 'plus one.');
   add_by_index(aggr, 6, asp(aggr_with_data[1], ?), IX_NVLD, aggr);
   verdict;

END_PROCEDURE; -- atc_add_by_index_list_entity
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of entity instances. The aggregate shall be empty.

aggr_with_data: (input) An aggregate containing entity instances that will be added to **aggr**.

6.2.82 atc_get_by_index_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get by index* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atc_get_by_index_list_entity(aggr : aggregate_instance;
     aggr_with_data : BAG [0:?] OF application_instance);
   LOCAL
      p : primitive;
   END_LOCAL;

   atc('atc_get_by_index_list_entity');

   purpose('Ensures that get_by_index reports an IX_NVLD error when '
      + 'the index submitted is outside of the legal range.');
   p := get_by_index(aggr, 0, IX_NVLD, aggr);
   p := get_by_index(aggr, 5, IX_NVLD, aggr);
   verdict;

   purpose('Ensures that get_by_index works correctly when the '
      + 'index submitted is from the legal range.');
   p := get_by_index(aggr, 1, NO_ERROR, ?);
   assert(p :=: asp(aggr_with_data[1], ?));
   p := get_by_index(aggr, 2, NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   assert(p :=: asp(aggr_with_data[2], ?));
   p := get_by_index(aggr, 3, NO_ERROR, ?);
   assert(p :=: asp(aggr_with_data[3], ?));
   p := get_by_index(aggr, 4, NO_ERROR, ?);
   assert(p :=: asp(aggr_with_data[4], ?));
   verdict;

END_PROCEDURE; -- atc_get_by_index_list_entity
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of entity instances. The aggregate shall be that processed by atc_add_by_index_list_entity.

aggr_with_data: (input) An aggregate containing entity instances that will be added to **aggr**.

6.2.83 atc_put_by_index_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *put by index* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atc_put_by_index_list_entity(aggr : aggregate_instance;
      inst : application_instance);
   LOCAL
      p : primitive;
   END_LOCAL;

   atc('atc_put_by_index_list_entity');

   purpose('Ensures that put_by_index reports an IX_NVLD error when '
      + 'the index submitted is outside of the legal range.');
   put_by_index(aggr, 0, asp(inst, ?), IX_NVLD, aggr);
   put_by_index(aggr, 5, asp(inst, ?), IX_NVLD, aggr);
   verdict;

   purpose('Ensures that put_by_index reports a VA_NSET error when an '
      + 'unset value is submitted.');
   put_by_index(aggr, 3, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_by_index reports a VT_NVLD error when value '
      + 'of a wrong type is submitted.');
   put_by_index(aggr, 3, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_by_index works correctly when all parameters '
      + 'are correct.');
   put_by_index(aggr, 3, asp(inst, ?), NO_ERROR, ?);
   p := get_by_index(aggr, 3, NO_ERROR, ?);
   assert(p :=: asp(inst, ?));
   verdict;

END_PROCEDURE; -- atc_put_by_index_list_entity
(*
```

Argument definitions:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

aggr: (input) An aggregate of type LIST of entity instances. The aggregate shall be that processed by atc_add_by_index_list_entity.

inst: (input) An entity instance.

6.2.84 atc_is_member_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *is member* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atc_is_member_list_entity(aggr : aggregate_instance;
    aggr_with_data : BAG [0:?] OF application_instance);
  LOCAL
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_is_member_list_entity');

  purpose('Ensures that is_member returns TRUE when value '
    + 'submitted belongs to the aggregate.');
  bool := is_member(aggr, asp(aggr_with_data[4], ?), NO_ERROR, ?);
  assert(bool);
  verdict;

  purpose('Ensures that is_member returns FALSE when value submitted '
    + 'does not belong to the aggregate.');
  bool := is_member(aggr, asp(aggr_with_data[3], ?), NO_ERROR, ?);
  assert(NOT bool);
  verdict;

END_PROCEDURE; -- atc_is_member_list_entity
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of entity instances. The aggregate shall be that processed by atc_put_by_index_list_entity.

aggr_with_data: (input) An aggregate containing entity instances that will be checked for inclusion in **aggr**.

6.2.85 atc_remove_by_index_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove by index* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_by_index_list_entity (aggr : aggregate_instance;
    inst : application_instance);
  LOCAL
    p : primitive;
  END_LOCAL;

  atc('atc_remove_by_index_list_entity');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that remove_by_index reports an IX_NVLD error when '
     + 'the index submitted is outside of the legal range.');
   remove_by_index(aggr, 0, IX_NVLD, aggr);
   remove_by_index(aggr, 5, IX_NVLD, aggr);
   verdict;

   purpose('Ensures that remove_by_index works correctly when the '
     + 'index submitted is from a legal range.');
   remove_by_index(aggr, 2, NO_ERROR, ?);
   p := get_by_index(aggr, 2, NO_ERROR, ?);
   assert(p :=: asp(inst, ?));
   verdict;

END_PROCEDURE; -- atc_remove_by_index_list_entity
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of entity instances. The aggregate shall be that processed by atc_put_by_index_list_entity.

inst: (input) An entity instance.

6.2.86 atc_add_before_current_member_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add before current member* for an aggregate of type LIST of entity instances and the SDAI operation *add after current member* for an empty aggregate.

EXPRESS specification:

```
*)
PROCEDURE atc_add_before_current_member_list_entity (iter : iterator;
     aggr_with_data : BAG [0:?] OF application_instance);
   LOCAL
     bool : BOOLEAN;
   END_LOCAL;

   atc('atc_add_before_current_member_list_entity');

   purpose('Ensures that after add_after_current_member operation for '
     + 'an empty list iterator is positioned at the beginning of the '
     + 'list (there is no current member).');
   atEnd(iter, NO_ERROR, ?);
   add_after_current_member(iter, asp(aggr_with_data[1], ?), NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   assert(bool);
   bool := remove_current_member(iter, NO_ERROR, ?);
   verdict;

   purpose('Ensures that add_before_current_member reports an IR_NEXS '
     + 'error when iterator is not provided.');
   add_before_current_member(?, asp(aggr_with_data[3], ?), IR_NEXS, ?);
   verdict;

   purpose('Ensures that add_before_current_member reports a VA_NSET '
     + 'error when an unset value is submitted.');
   add_before_current_member(iter, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that add_before_current_member reports a VT_NVLD '
     + 'error when value of a wrong type is submitted.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   add_before_current_member(iter, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that after add_before_current_member operation for '
      + 'an empty list iterator is positioned at the end of the list '
      + '(there is no current member).');
   beginning(iter, NO_ERROR, ?);
   add_before_current_member(iter, asp(aggr_with_data[3], ?), NO_ERROR, ?);
   bool := previous(iter, NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that add_before_current_member works correctly when '
      + 'all parameters are correct.');
   add_before_current_member(iter, asp(aggr_with_data[2], ?), NO_ERROR, ?);
   atEnd(iter, NO_ERROR, ?);
   add_before_current_member(iter, asp(aggr_with_data[4], ?), NO_ERROR, ?);
   verdict;

END_PROCEDURE; -- atc_add_before_current_member_list_entity
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of entity instances.

aggr_with_data: (input) An aggregate containing entity instances that will be added to the list specified by the iterator. The aggregate shall be empty.

6.2.87 atc_add_after_current_member_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add after current member* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atc_add_after_current_member_list_entity (iter : iterator;
     aggr_with_data : BAG [0:?] OF application_instance);

   atc('atc_add_after_current_member_list_entity');

   (* Positioning of the iterator at the end of the aggregate. *)
   atEnd(iter, NO_ERROR, ?);

   purpose('Ensures that add_after_current_member reports an IR_NEXS '
      + 'error when iterator is not provided.');
   add_after_current_member(?, asp(aggr_with_data[5], ?), IR_NEXS, ?);
   verdict;

   purpose('Ensures that add_after_current_member reports a VA_NSET '
      + 'error when an unset value is submitted.');
   add_after_current_member(iter, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that add_after_current_member reports a VT_NVLD '
      + 'error when value of a wrong type is submitted.');
   add_after_current_member(iter, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that add_after_current_member works correctly when '
      + 'all parameters are correct.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   add_after_current_member(iter, asp(aggr_with_data[5], ?), NO_ERROR, ?);
   add_after_current_member(iter, asp(aggr_with_data[6], ?), NO_ERROR, ?);
   beginning(iter, NO_ERROR, ?);
   add_after_current_member(iter, asp(aggr_with_data[1], ?), NO_ERROR, ?);
   verdict;

END_PROCEDURE; -- atc_add_after_current_member_list_entity
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of entity instances.

aggr_with_data: (input) An aggregate containing entity instances that will be added to the list specified by the iterator. The aggregate shall be that processed by atc_add_before_current_member_list_entity.

6.2.88 atc_get_current_member_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get current member* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atc_get_current_member_list_entity
    (iter : iterator; aggr_with_data : BAG [0:?] OF application_instance);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_get_current_member_list_entity');

   purpose('Ensures that get_current_member reports an IR_NEXS error '
      + 'when iterator is not provided.');
   p := get_current_member(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that get_current_member reports an IR_NSET error '
      + 'when iterator has no current member set.');
   beginning(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
   atEnd(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
   verdict;

   purpose('Ensures that get_current_member works correctly when '
      + 'iterator has current member set.');
   bool := previous(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p :=: asp(aggr_with_data[6], ?));
   bool := previous(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p :=: asp(aggr_with_data[5], ?));
   bool := previous(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p :=: asp(aggr_with_data[4], ?));
   bool := previous(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p :=: asp(aggr_with_data[3], ?));
   bool := previous(iter, NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p :=: asp(aggr_with_data[2], ?));
   bool := previous(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p :=: asp(aggr_with_data[1], ?));
   verdict;

END_PROCEDURE; -- atc_get_current_member_list_entity
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of entity instances.

aggr_with_data: (input) An aggregate containing entity instances that will be compared with elements in the list specified by the iterator. The aggregate shall be that processed by atc_add_after_current_member_list_entity.

6.2.89 atc_put_current_member_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *put current member* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atc_put_current_member_list_entity
      (iter : iterator; inst : application_instance);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_put_current_member_list_entity');

   purpose('Ensures that put_current_member reports an IR_NEXS error '
      + 'when iterator is not provided.');
   put_current_member(?, asp(inst, ?), IR_NEXS, ?);
   verdict;

   purpose('Ensures that put_current_member reports an IR_NSET error '
      + 'when iterator has no current member set.');
   beginning(iter, NO_ERROR, ?);
   put_current_member(iter, asp(inst, ?), IR_NSET, iter);
   atEnd(iter, NO_ERROR, ?);
   put_current_member(iter, asp(inst, ?), IR_NSET, iter);
   verdict;

   purpose('Ensures that put_current_member reports a VA_NSET error '
      + 'when an unset value is submitted.');
   put_current_member(iter, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_current_member reports a VT_NVLD error '
      + 'when value of a wrong type is submitted.');
   put_current_member(iter, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_current_member works correctly '
      + 'when all parameters are correct.');
   bool := previous(iter, NO_ERROR, ?);
   put_current_member(iter, asp(inst, ?), NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p :=: asp(inst, ?));
   verdict;

END_PROCEDURE; -- atc_put_current_member_list_entity
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of entity instances. The aggregate shall be that processed by atc_add_after_current_member_list_entity.

inst: (input) An entity instance.

6.2.90 atc_remove_current_member_list_entity

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove current member* for an aggregate of type LIST of entity instances.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_current_member_list_entity
      (iter : iterator; inst : application_instance);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_remove_current_member_list_entity');

   purpose('Ensures that remove_current_member reports an IR_NEXS '
      + 'error when iterator is not provided.');
   bool := remove_current_member(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that remove_current_member reports an IR_NSET '
      + 'error when iterator has no current member set.');
   atEnd(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, IR_NSET, iter);
   beginning(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, IR_NSET, iter);
   verdict;

   purpose('Ensures that remove_current_member works correctly '
      + 'when iterator has current member set.');
   bool := next(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, NO_ERROR, ?);
   assert(bool);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p :=: asp(inst, ?));
   verdict;

   purpose('Ensures that remove_current_member returns FALSE when '
      + 'iterator refers to the last member of the aggregate.');
   atEnd(iter, NO_ERROR, ?);
   bool := previous(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_remove_current_member_list_entity
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of entity instances. The aggregate shall be that processed by atc_put_current_member_list_entity.

inst: (input) An entity instance.

6.2.91 atg_set_logical

This abstract test group shall be used for the assessment of the implementation of the SDAI operations *add unordered*, *remove unordered*, *is member*, *get current member*, *put current member*, and *remove current member* for an aggregate of type SET of LOGICAL.

EXPRESS specification:

```
*)
PROCEDURE atg_set_logical(aggr : aggregate_instance; iter : iterator);

  (* Testing of the aggregate operation add unordered. *)
  atc_add_unordered_set_logical(aggr, iter);

  (* Testing of the aggregate operation remove unordered. *)
  atc_remove_unordered_set_logical(aggr);

  (* Testing of the aggregate operation is member. *)
  atc_is_member_set_logical(aggr);

  (* Making the given set empty. *)
  macro_clear_aggregate(aggr);

  (* Initializing the set with all three logical values. *)
  add_unordered(aggr, asp(UNKNOWN, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(FALSE, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(TRUE, ?), NO_ERROR, ?);

  (* Testing of the aggregate operation get current member. *)
  atc_get_current_member_set_logical(iter);

  (* Testing of the aggregate operation remove current member. *)
  atc_remove_current_member_set_logical(aggr, iter);

  (* Making the given set empty. *)
  macro_clear_aggregate(aggr);

  (* Initializing the set with logical values UNKNOWN and TRUE. *)
  add_unordered(aggr, asp(UNKNOWN, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(TRUE, ?), NO_ERROR, ?);

  (* Testing of the aggregate operation put current member. *)
  atc_put_current_member_set_logical(aggr, iter);

END_PROCEDURE; -- atg_set_logical
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of LOGICAL. The aggregate shall be empty.

iter: (input) An iterator over an aggregate specified by the first argument.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.92 atc_add_unordered_set_logical

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add unordered* for an aggregate of type SET of LOGICAL.

EXPRESS specification:

```
*)
PROCEDURE atc_add_unordered_set_logical
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    aggr_auxiliary : SET [0:?] OF primitive;
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_add_unordered_set_logical');

  purpose('Ensures that add_unordered reports a VA_NSET error when '
    + 'an unset value is submitted.');
  add_unordered(aggr, ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that add_unordered reports a VT_NVLD error when '
    + 'value of a wrong type is submitted.');
  add_unordered(aggr, asp('something', ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that add_unordered works correctly when all '
    + 'parameters are correct.');
  add_unordered(aggr, asp(UNKNOWN, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(FALSE, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(TRUE, ?), NO_ERROR, ?);
  beginning(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[1] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[2] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[3] := get_current_member(iter, NO_ERROR, ?);
  bool := macro_compare_aggregates(aggr, aggr_auxiliary);
  assert(bool);
  verdict;

END_PROCEDURE; -- atc_add_unordered_set_logical
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of LOGICAL. The aggregate shall be empty.

iter: (input) An iterator over **aggr**.

6.2.93 atc_remove_unordered_set_logical

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove unordered* for an aggregate of type SET of LOGICAL.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_unordered_set_logical (aggr : aggregate_instance);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   atc('atc_remove_unordered_set_logical');

   purpose('Ensures that remove_unordered reports a VA_NSET error '
      + 'when an unset value is submitted.');
   remove_unordered(aggr, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that remove_unordered reports a VT_NVLD error '
      + 'when value of a wrong type is submitted.');
   remove_unordered(aggr, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that remove_unordered works correctly '
      + 'when parameters are correct.');
   remove_unordered(aggr, asp(FALSE, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that remove_unordered reports a VA_NEXS error '
      + 'when value does not exist in the aggregate.');
   remove_unordered(aggr, asp(FALSE, ?), VA_NEXS, ?);
   verdict;

END_PROCEDURE; -- atc_remove_unordered_set_logical
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of LOGICAL. The aggregate shall be that processed by atc_add_unordered_set_logical.

6.2.94 atc_is_member_set_logical

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *is member* and *get member count* for an aggregate of type SET of LOGICAL.

EXPRESS specification:

```
*)
PROCEDURE atc_is_member_set_logical(aggr : aggregate_instance);
   LOCAL
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_is_member_set_logical');

   purpose('Ensures that is_member returns TRUE when value '
      + 'submitted belongs to the aggregate.');
   bool := is_member(aggr, asp(UNKNOWN, ?), NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that is_member returns FALSE when value submitted '
      + 'does not belong to the aggregate.');
   bool := is_member(aggr, asp(FALSE, ?), NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_member_count works correctly.');
   assert(get_member_count(aggr, NO_ERROR, ?) = 2);
   verdict;

END_PROCEDURE; -- atc_is_member_set_logical
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of LOGICAL. The aggregate shall be that processed by atc_remove_unordered_set_logical.

6.2.95 atc_get_current_member_set_logical

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get current member* for an aggregate of type SET of LOGICAL.

EXPRESS specification:

```
*)
PROCEDURE atc_get_current_member_set_logical(iter : iterator);
   LOCAL
      aggr : SET [0:?] OF primitive;
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_get_current_member_set_logical');

   purpose('Ensures that get_current_member reports an IR_NEXS error '
      + 'when iterator is not provided.');
   p := get_current_member(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that get_current_member reports an IR_NSET error '
      + 'when iterator is at the beginning.');
   beginning(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
   verdict;

   purpose('Ensures that get_current_member works correctly when '
      + 'iterator has current member set.');
   bool := next(iter, NO_ERROR, ?);
   aggr[1] := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr[2] := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr[3] := get_current_member(iter, NO_ERROR, ?);
   bool := macro_compare_aggregates(aggr, [UNKNOWN, FALSE, TRUE]);
   assert(bool);
   verdict;

   purpose('Ensures that get_current_member reports an IR_NSET error '
      + 'when iterator is at the end.');
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
   verdict;

END_PROCEDURE; -- atc_get_current_member_set_logical
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type SET of LOGICAL. The aggregate shall contain values UNKNOWN, FALSE and TRUE.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.96 atc_put_current_member_set_logical

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *put current member* for an aggregate of type SET of LOGICAL.

EXPRESS specification:

```
*)
PROCEDURE atc_put_current_member_set_logical
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_put_current_member_set_logical');

   purpose('Ensures that put_current_member reports an IR_NEXS error '
      + 'when iterator is not provided.');
   put_current_member(?, asp(FALSE, ?), IR_NEXS, ?);
   verdict;

   purpose('Ensures that put_current_member reports an IR_NSET error '
      + 'when iterator is at the beginning.');
   beginning(iter, NO_ERROR, ?);
   put_current_member(iter, asp(FALSE, ?), IR_NSET, iter);
   verdict;

   purpose('Ensures that put_current_member reports a VA_NSET error '
      + 'when an unset value is submitted.');
   bool := next(iter, NO_ERROR, ?);
   put_current_member(iter, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_current_member reports a VT_NVLD error '
      + 'when value of a wrong type is submitted.');
   put_current_member(iter, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_current_member works correctly '
      + 'when all parameters are correct.');
   p := get_current_member(iter, NO_ERROR, ?);
   put_current_member(iter, asp(FALSE, ?), NO_ERROR, ?);
   bool := is_member(aggr, asp(FALSE, ?), NO_ERROR, ?);
   assert(bool);
   bool := is_member(aggr, p, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that put_current_member reports an IR_NSET error '
      + 'when iterator is at the end.');
   bool := next(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   put_current_member(iter, asp(FALSE, ?), IR_NSET, iter);
   verdict;

END_PROCEDURE; -- atc_put_current_member_set_logical
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of LOGICAL. Its elements shall be UNKNOWN and TRUE.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.97 atc_remove_current_member_set_logical

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove current member* for an aggregate of type SET of LOGICAL.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_current_member_set_logical
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_remove_current_member_set_logical');

  purpose('Ensures that remove_current_member reports an IR_NEXS '
    + 'error when iterator is not provided.');
  bool := remove_current_member(?, IR_NEXS, ?);
  verdict;

  purpose('Ensures that remove_current_member reports an IR_NSET '
    + 'error when iterator is at the beginning.');
  beginning(iter, NO_ERROR, ?);
  bool := remove_current_member(iter, IR_NSET, iter);
  verdict;

  purpose('Ensures that remove_current_member works correctly '
    + 'when iterator has current member set.');
  bool := next(iter, NO_ERROR, ?);
  p := get_current_member(iter, NO_ERROR, ?);
  bool := remove_current_member(iter, NO_ERROR, ?);
  assert(bool);
  bool := is_member(aggr, p, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that remove_current_member returns FALSE when '
    + 'iterator refers to the last member of the aggregate.');
  bool := next(iter, NO_ERROR, ?);
  bool := remove_current_member(iter, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that remove_current_member reports an IR_NSET '
    + 'error when iterator is at the end.');
  bool := remove_current_member(iter, IR_NSET, iter);
  verdict;

END_PROCEDURE; -- atc_remove_current_member_set_logical
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of LOGICAL. Its elements shall be UNKNOWN, FALSE and TRUE.

iter: (input) An iterator over an aggregate specified by the first argument.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.98 atg_set_simple_defined_type

This abstract test group shall be used for the assessment of the implementation of the SDAI operation *add unordered*, *remove unordered*, *is member*, *get current member*, *put current member*, and *remove current member* for an aggregate of type SET of EXPRESS TYPE.

EXPRESS specification:

```
*)
PROCEDURE atg_set_simple_defined_type
      (aggr : aggregate_instance; iter : iterator);

   (* Testing of the aggregate operation add unordered. *)
   atc_add_unordered_set_simple_defined_type(aggr, iter);

   (* Testing of the aggregate operation remove unordered. *)
   atc_remove_unordered_set_simple_defined_type(aggr);

   (* Testing of the aggregate operation is member. *)
   atc_is_member_set_simple_defined_type(aggr);

   (* Making the given set empty. *)
   macro_clear_aggregate(aggr);

   (* Initializing the set with three integer values. *)
   add_unordered(aggr, asp(10, ?), NO_ERROR, ?);
   add_unordered(aggr, asp(20, ?), NO_ERROR, ?);
   add_unordered(aggr, asp(30, ?), NO_ERROR, ?);

   (* Testing of the aggregate operation get current member. *)
   atc_get_current_member_set_simple_defined_type(iter);

   (* Testing of the aggregate operation remove current member. *)
   atc_remove_current_member_set_simple_defined_type(aggr, iter);

   (* Making the given set empty. *)
   macro_clear_aggregate(aggr);

   (* Initializing the set with integer values 10 and 30. *)
   add_unordered(aggr, asp(10, ?), NO_ERROR, ?);
   add_unordered(aggr, asp(30, ?), NO_ERROR, ?);

   (* Testing of the aggregate operation put current member. *)
   atc_put_current_member_set_simple_defined_type(aggr, iter);

END_PROCEDURE; -- atg_set_simple_defined_type
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of EXPRESS TYPE, specifically of type xi in ESTS schema. The aggregate shall be empty.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.99 atc_add_unordered_set_simple_defined_type

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add unordered* for an aggregate of type SET of EXPRESS TYPE.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
*)
PROCEDURE atc_add_unordered_set_simple_defined_type
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    aggr_auxiliary : SET [0:?] OF primitive;
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_add_unordered_set_simple_defined_type');

  purpose('Ensures that add_unordered reports a VA_NSET error '
    + 'when an unset value is submitted.');
  add_unordered(aggr, ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that add_unordered reports a VT_NVLD error '
    + 'when value of a wrong type is submitted.');
  add_unordered(aggr, asp('something', ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that add_unordered works correctly when all '
    + 'parameters are correct.');
  add_unordered(aggr, asp(10, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(20, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(30, ?), NO_ERROR, ?);
  beginning(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[1] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[2] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[3] := get_current_member(iter, NO_ERROR, ?);
  bool := macro_compare_aggregates(aggr, aggr_auxiliary);
  assert(bool);
  verdict;

END_PROCEDURE; -- atc_add_unordered_set_simple_defined_type
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of EXPRESS TYPE. The aggregate shall be empty.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.100 atc_remove_unordered_set_simple_defined_type

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove unordered* for an aggregate of type SET of EXPRESS TYPE.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_unordered_set_simple_defined_type
      (aggr : aggregate_instance);

  atc('atc_remove_unordered_set_simple_defined_type');

  purpose('Ensures that remove_unordered reports a VA_NSET error '
      + 'when an unset value is submitted.');
  remove_unordered(aggr, ?, VA_NSET, ?);
  verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that remove_unordered reports a VT_NVLD error '
     + 'when value of a wrong type is submitted.');
   remove_unordered(aggr, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that remove_unordered works correctly '
     + 'when parameters are correct.');
   remove_unordered(aggr, asp(20, ?), NO_ERROR, ?);
   verdict;

   purpose('Ensures that remove_unordered reports a VA_NEXS error '
     + 'when value does not exist in the aggregate.');
   remove_unordered(aggr, asp(20, ?), VA_NEXS, ?);
   verdict;

END_PROCEDURE; -- atc_remove_unordered_set_simple_defined_type
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of EXPRESS TYPE. The aggregate shall be that processed by atc_add_unordered_set_simple_defined_type.

6.2.101 atc_is_member_set_simple_defined_type

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *is member* for an aggregate of type SET of EXPRESS TYPE.

EXPRESS specification:

```
*)
PROCEDURE atc_is_member_set_simple_defined_type
      (aggr : aggregate_instance);
   LOCAL
     bool : BOOLEAN;
   END_LOCAL;

   atc('atc_is_member_set_simple_defined_type');

   purpose('Ensures that is_member returns TRUE when value '
     + 'submitted belongs to the aggregate.');
   bool := is_member(aggr, asp(10, ?), NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that is_member returns FALSE when value '
     + 'submitted does not belong to the aggregate.');
   bool := is_member(aggr, asp(20, ?), NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_is_member_set_simple_defined_type
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of EXPRESS TYPE. The aggregate shall be that processed by atc_remove_unordered_set_simple_defined_type.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

6.2.102 atc_get_current_member_set_simple_defined_type

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get current member* for an aggregate of type SET of EXPRESS TYPE.

EXPRESS specification:

```
*)
PROCEDURE atc_get_current_member_set_simple_defined_type
      (iter : iterator);
  LOCAL
    aggr : SET [0:?] OF primitive;
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_get_current_member_set_simple_defined_type');

  purpose('Ensures that get_current_member reports an IR_NEXS error '
    + 'when iterator is not provided.');
  p := get_current_member(?, IR_NEXS, ?);
  verdict;

  purpose('Ensures that get_current_member reports an IR_NSET error '
    + 'when iterator is at the beginning.');
  beginning(iter, NO_ERROR, ?);
  p := get_current_member(iter, IR_NSET, iter);
  verdict;

  purpose('Ensures that get_current_member works correctly '
    + 'when iterator has current member set.');
  bool := next(iter, NO_ERROR, ?);
  aggr[1] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr[2] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr[3] := get_current_member(iter, NO_ERROR, ?);
  bool := macro_compare_aggregates(aggr, [10, 20, 30]);
  assert(bool);
  verdict;

  purpose('Ensures that get_current_member reports an IR_NSET error '
    + 'when iterator is at the end.');
  bool := next(iter, NO_ERROR, ?);
  p := get_current_member(iter, IR_NSET, iter);
  verdict;

END_PROCEDURE; -- atc_get_current_member_set_simple_defined_type
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Argument definitions:

iter: (input) An iterator over an aggregate of type SET of EXPRESS TYPE. The aggregate shall contain values 10, 20 and 30.

6.2.103 atc_put_current_member_set_simple_defined_type

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *put current* member for an aggregate of type SET of EXPRESS TYPE.

EXPRESS specification:

```
*)
PROCEDURE atc_put_current_member_set_simple_defined_type
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_put_current_member_set_simple_defined_type');

  purpose('Ensures that put_current_member reports an IR_NEXS error '
    + 'when iterator is not provided.');
  put_current_member(?, asp(20, ?), IR_NEXS, ?);

  purpose('Ensures that put_current_member reports an IR_NSET error '
    + 'when iterator is at the beginning.');
  beginning(iter, NO_ERROR, ?);
  put_current_member(iter, asp(20, ?), IR_NSET, iter);
  verdict;

  purpose('Ensures that put_current_member reports a VA_NSET error '
    + 'when an unset value is submitted.');
  bool := next(iter, NO_ERROR, ?);
  put_current_member(iter, ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that put_current_member reports a VT_NVLD error '
    + 'when value of a wrong type is submitted.');
  put_current_member(iter, asp('something', ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that put_current_member works correctly '
    + 'when all parameters are correct.');
  p := get_current_member(iter, NO_ERROR, ?);
  put_current_member(iter, asp(30, ?), NO_ERROR, ?);
  bool := is_member(aggr, asp(30, ?), NO_ERROR, ?);
  assert(bool);
  bool := is_member(aggr, p, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that put_current_member reports an IR_NSET error '
    + 'when iterator is at the end.');
  bool := next(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  put_current_member(iter, asp(20, ?), IR_NSET, iter);
  verdict;

END_PROCEDURE; -- atc_put_current_member_set_simple_defined_type
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of EXPRESS TYPE xi. Its elements shall be 10 and 30.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.104 atc_remove_current_member_set_simple_defined_type

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove current member* for an aggregate of type SET of EXPRESS TYPE.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
*)
PROCEDURE atc_remove_current_member_set_simple_defined_type
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_remove_current_member_set_simple_defined_type');

  purpose('Ensures that remove_current_member reports an IR_NEXS '
    + 'error when iterator is not provided.');
  bool := remove_current_member(?, IR_NEXS, ?);
  verdict;

  purpose('Ensures that remove_current_member reports an IR_NSET '
    + 'error when iterator is at the beginning.');
  beginning(iter, NO_ERROR, ?);
  bool := remove_current_member(iter, IR_NSET, iter);
  verdict;

  purpose('Ensures that remove_current_member works correctly '
    + 'when iterator has current member set.');
  bool := next(iter, NO_ERROR, ?);
  p := get_current_member(iter, NO_ERROR, ?);
  bool := remove_current_member(iter, NO_ERROR, ?);
  assert(bool);
  bool := is_member(aggr, p, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that remove_current_member returns FALSE when '
    + 'iterator refers to the last member of the aggregate.');
  bool := next(iter, NO_ERROR, ?);
  bool := remove_current_member(iter, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that remove_current_member reports an IR_NSET '
    + 'error when iterator is at the end.');
  bool := remove_current_member(iter, IR_NSET, iter);
  verdict;

END_PROCEDURE; -- atc_remove_current_member_set_simple_defined_type
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of EXPRESS TYPE. Its elements shall be 10, 20 and 30.

iter: (input) An iterator over an aggregate specified by the first argument.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.105 atg_bag_of_string

This abstract test group shall be used for the assessment of the implementation of the SDAI operations *add unordered*, *remove unordered*, *is member*, *get current member*, *put current member*, and *remove current member* for an aggregate of type BAG of STRING.

EXPRESS specification:

```
*)
PROCEDURE atg_bag_of_string(aggr : aggregate_instance; iter : iterator);
```

```
  (* Making the given bag empty. *)
  macro_clear_aggregate(aggr);

  (* Testing of the aggregate operation add unordered. *)
  atc_add_unordered_bag_of_string(aggr, iter);

  (* Testing of the aggregate operation remove unordered. *)
  atc_remove_unordered_bag_of_string(aggr, iter);

  (* Testing of the aggregate operation is member. *)
  atc_is_member_bag_of_string(aggr);

  (* Making the given bag empty. *)
  macro_clear_aggregate(aggr);

  (* Initializing the bag with three STRING values. *)
  add_unordered(aggr, asp('first', ?), NO_ERROR, ?);
  add_unordered(aggr, asp('second', ?), NO_ERROR, ?);
  add_unordered(aggr, asp('second', ?), NO_ERROR, ?);

  (* Testing of the aggregate operation get current member. *)
  atc_get_current_member_bag_of_string(aggr, iter);

  (* Testing of the aggregate operation remove current member. *)
  atc_remove_current_member_bag_of_string(aggr, iter);

  (* Making the given bag empty. *)
  macro_clear_aggregate(aggr);

  (* Initializing the bag with two STRING values. *)
  add_unordered(aggr, asp('first', ?), NO_ERROR, ?);
  add_unordered(aggr, asp('second', ?), NO_ERROR, ?);

  (* Testing of the aggregate operation put current member. *)
  atc_put_current_member_bag_of_string(aggr, iter);

END_PROCEDURE; -- atg_bag_of_string
(*
```

<ins>Argument definitions:</ins>

aggr: (input) An aggregate of type BAG of STRING.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.106 atc_add_unordered_bag_of_string

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add unordered* for an aggregate of type BAG of STRING.

<ins>EXPRESS specification:</ins>

```
*)
PROCEDURE atc_add_unordered_bag_of_string
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    aggr_auxiliary : BAG [0:?] OF primitive;
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_add_unordered_bag_of_string');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   purpose('Ensures that add_unordered reports a VA_NSET error '
      + 'when an unset value is submitted.');
   add_unordered(aggr, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that add_unordered reports a VT_NVLD error '
      + 'when value of a wrong type is submitted.');
   add_unordered(aggr, asp(5.5, ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that add_unordered works correctly '
      + 'when all parameters are correct.');
   add_unordered(aggr, asp('first', ?), NO_ERROR, ?);
   add_unordered(aggr, asp('second', ?), NO_ERROR, ?);
   add_unordered(aggr, asp('second', ?), NO_ERROR, ?);
   beginning(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr_auxiliary[1] := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr_auxiliary[2] := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr_auxiliary[3] := get_current_member(iter, NO_ERROR, ?);
   bool := macro_compare_aggregates(aggr, aggr_auxiliary);
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_add_unordered_bag_of_string
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of STRING. The aggregate shall be empty.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.107 atc_remove_unordered_bag_of_string

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove unordered* for an aggregate of type BAG of STRING.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_unordered_bag_of_string
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p1 : primitive;
      p2 : primitive;
   END_LOCAL;

   atc('atc_remove_unordered_bag_of_string');

   purpose('Ensures that remove_unordered reports a VA_NSET error '
      + 'when an unset value is submitted.');
   remove_unordered(aggr, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that remove_unordered reports a VT_NVLD error '
      + 'when value of a wrong type is submitted.');
   remove_unordered(aggr, asp(5.5, ?), VT_NVLD, ?);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that remove_unordered works correctly when '
      + 'parameters are correct and removes only one occurrence '
      + 'of the specified value.');
   remove_unordered(aggr, asp('second', ?), NO_ERROR, ?);
   beginning(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   p1 := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   p2 := get_current_member(iter, NO_ERROR, ?);
   bool := macro_compare_aggregates([p1, p2], ['first', 'second']);
   assert(bool);
   verdict;

   purpose('Ensures that remove_unordered reports a VA_NEXS error '
      + 'when value does not exist in the aggregate.');
   remove_unordered(aggr, asp('third', ?), VA_NEXS, ?);
   verdict;

END_PROCEDURE; -- atc_remove_unordered_bag_of_string
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of STRING. The aggregate shall be that processed by atc_add_unordered_bag_of_string.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.108 atc_is_member_bag_of_string

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *is member* and *get member count* for an aggregate of type BAG of STRING.

EXPRESS specification:

```
*)
PROCEDURE atc_is_member_bag_of_string(aggr : aggregate_instance);
   LOCAL
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_is_member_bag_of_string');

   purpose('Ensures that is_member returns TRUE when value '
      + 'submitted belongs to the aggregate.');
   bool := is_member(aggr, asp('first', ?), NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that is_member returns FALSE when value submitted '
      + 'does not belong to the aggregate.');
   bool := is_member(aggr, asp('third', ?), NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_member_count works correctly.');
   assert(get_member_count(aggr, NO_ERROR, ?) = 2);
   verdict;

END_PROCEDURE; -- atc_is_member_bag_of_string
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Argument definitions:

aggr: (input) An aggregate of type BAG of STRING. The aggregate shall be that processed by atc_remove_unordered_bag_of_string.

6.2.109 atc_get_current_member_bag_of_string

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get current member* for an aggregate of type BAG of STRING.

EXPRESS specification:

```
*)
PROCEDURE atc_get_current_member_bag_of_string
     (aggr : aggregate_instance; iter : iterator);
  LOCAL
    aggr_auxiliary : BAG [0:?] OF primitive;
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_get_current_member_bag_of_string');

  purpose('Ensures that get_current_member reports an IR_NEXS error '
    + 'when iterator is not provided.');
  p := get_current_member(?, IR_NEXS, ?);
  verdict;

  purpose('Ensures that get_current_member reports an IR_NSET error '
    + 'when iterator is at the beginning.');
  beginning(iter, NO_ERROR, ?);
  p := get_current_member(iter, IR_NSET, iter);
  verdict;

  purpose('Ensures that get_current_member works correctly when '
    + 'iterator has current member set.');
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[1] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[2] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[3] := get_current_member(iter, NO_ERROR, ?);
  bool := macro_compare_aggregates(aggr, aggr_auxiliary);
  assert(bool);
  verdict;

  purpose('Ensures that get_current_member reports an IR_NSET error '
    + 'when iterator is at the end.');
  bool := next(iter, NO_ERROR, ?);
  p := get_current_member(iter, IR_NSET, iter);
  verdict;

END_PROCEDURE; -- atc_get_current_member_bag_of_string
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of STRING. The aggregate shall contain three values of type STRING.

iter: (input) An iterator over **aggr**.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.110 atc_put_current_member_bag_of_string

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *put current member* for an aggregate of type BAG of STRING.

EXPRESS specification:

```
*)
PROCEDURE atc_put_current_member_bag_of_string
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_put_current_member_bag_of_string');

  purpose('Ensures that put_current_member reports an IR_NEXS error '
    + 'when iterator is not provided.');
  put_current_member(?, asp('third', ?), IR_NEXS, ?);
  verdict;

  purpose('Ensures that put_current_member reports an IR_NSET error '
    + 'when iterator is at the beginning.');
  beginning(iter, NO_ERROR, ?);
  put_current_member(iter, asp('third', ?), IR_NSET, iter);
  verdict;

  purpose('Ensures that put_current_member reports a VA_NSET error '
    + 'when an unset value is submitted.');
  bool := next(iter, NO_ERROR, ?);
  put_current_member(iter, ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that put_current_member reports a VT_NVLD error '
    + 'when value of a wrong type is submitted.');
  put_current_member(iter, asp(5.5, ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that put_current_member works correctly when '
    + 'all parameters are correct.');
  p := get_current_member(iter, NO_ERROR, ?);
  put_current_member(iter, asp('third', ?), NO_ERROR, ?);
  bool := is_member(aggr, asp('third', ?), NO_ERROR, ?);
  assert(bool);
  bool := is_member(aggr, p, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that put_current_member reports an IR_NSET error '
    + 'when iterator is at the end.');
  bool := next(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  put_current_member(iter, asp('third', ?), IR_NSET, iter);
  verdict;

END_PROCEDURE; -- atc_put_current_member_bag_of_string
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of STRING. Its elements shall be strings different than string 'third'.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

iter: (input) An iterator over **aggr**.

6.2.111 atc_remove_current_member_bag_of_string

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove current member* for an aggregate of type BAG of STRING.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_current_member_bag_of_string
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_remove_current_member_bag_of_string');

   purpose('Ensures that remove_current_member reports an IR_NEXS '
      + 'error when iterator is not provided.');
   bool := remove_current_member(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that remove_current_member reports an IR_NSET '
      + 'error when iterator is at the beginning.');
   beginning(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, IR_NSET, iter);
   verdict;

   purpose('Ensures that remove_current_member works correctly '
      + 'when iterator has current member set.');
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, NO_ERROR, ?);
   assert(bool);
   bool := is_member(aggr, p, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that remove_current_member returns FALSE when '
      + 'iterator refers to the last member of the aggregate.');
   bool := next(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that remove_current_member reports an IR_NSET '
      + 'error when iterator is at the end.');
   bool := remove_current_member(iter, IR_NSET, iter);
   verdict;

END_PROCEDURE; -- atc_remove_current_member_bag_of_string
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of STRING. The aggregate shall contain three values of type STRING, first of which is unique.

iter: (input) An iterator over an aggregate specified by the first argument.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.112 atg_bag_of_real

This abstract test group shall be used for the assessment of the implementation of the SDAI operations *add unordered*, *remove unordered*, *is member*, *get current member*, *put current member*, and *remove current member* for an aggregate of type BAG of REAL.

EXPRESS specification:

```
*)
PROCEDURE atg_bag_of_real(aggr : aggregate_instance; iter : iterator);

  (* Making the given bag empty. *)
  macro_clear_aggregate(aggr);

  (* Testing of the aggregate operation add unordered. *)
  atc_add_unordered_bag_of_real(aggr, iter);

  (* Testing of the aggregate operation remove unordered. *)
  remove_unordered_bag_of_real(aggr, iter);

  (* Testing of the aggregate operation is member. *)
  atc_is_member_bag_of_real(aggr);

  (* Making the given bag empty. *)
  macro_clear_aggregate(aggr);

  (* Initializing the bag with three REAL values. *)
  add_unordered(aggr, asp(1.1, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(2.2, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(2.2, ?), NO_ERROR, ?);

  (* Testing of the aggregate operation get current member. *)
  atc_get_current_member_bag_of_real(aggr, iter);

  (* Testing of the aggregate operation remove current member. *)
  atc_remove_current_member_bag_of_real(aggr, iter);

  (* Making the given bag empty. *)
  macro_clear_aggregate(aggr);

  (* Initializing the bag with two REAL values. *)
  add_unordered(aggr, asp(1.1, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(2.2, ?), NO_ERROR, ?);

  (* Testing of the aggregate operation put current member. *)
  atc_put_current_member_bag_of_real(aggr, iter);

END_PROCEDURE; -- atg_bag_of_real
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of REAL.
iter: (input) An iterator over an aggregate specified by the first argument.

6.2.113 atc_add_unordered_bag_of_real

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *add unordered* for an aggregate of type BAG of REAL.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
*)
PROCEDURE atc_add_unordered_bag_of_real
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    aggr_auxiliary : BAG [0:?] OF primitive;
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_add_unordered_bag_of_real');

  purpose('Ensures that add_unordered reports a VA_NSET error when '
    + 'an unset value is submitted.');
  add_unordered(aggr, ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that add_unordered reports a VT_NVLD error '
    + 'when value of a wrong type is submitted.');
  add_unordered(aggr, asp('something', ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that add_unordered works correctly when all '
    + 'parameters are correct.');
  add_unordered(aggr, asp(1.1, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(2.2, ?), NO_ERROR, ?);
  add_unordered(aggr, asp(2.2, ?), NO_ERROR, ?);
  beginning(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[1] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[2] := get_current_member(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  aggr_auxiliary[3] := get_current_member(iter, NO_ERROR, ?);
  bool := macro_compare_aggregates(aggr, aggr_auxiliary);
  assert(bool);
  verdict;

END_PROCEDURE; -- atc_add_unordered_bag_of_real
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of REAL. The aggregate shall be empty.

iter: (input) An iterator over **aggr**.

6.2.114 atc_remove_unordered_bag_of_real

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove unordered* for an aggregate of type BAG of REAL.

EXPRESS specification:

```
*)
PROCEDURE remove_unordered_bag_of_real
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    bool : BOOLEAN;
    p1 : primitive;
    p2 : primitive;
  END_LOCAL;

  atc('remove_unordered_bag_of_real');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that remove_unordered reports a VA_NSET error '
      + 'when an unset value is submitted.');
   remove_unordered(aggr, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that remove_unordered reports a VT_NVLD '
      + 'error when value of a wrong type is submitted.');
   remove_unordered(aggr, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that remove_unordered works correctly when '
      + 'parameters are correct and removes only one occurrence '
      + 'of the specified value.');
   remove_unordered(aggr, asp(2.2, ?), NO_ERROR, ?);
   beginning(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   p1 := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   p2 := get_current_member(iter, NO_ERROR, ?);
   bool := macro_compare_aggregates([p1, p2], [1.1, 2.2]);
   assert(bool);
   verdict;

   purpose('Ensures that remove_unordered reports a VA_NEXS error '
      + 'when value does not exist in the aggregate.');
   remove_unordered(aggr, asp(3.3, ?), VA_NEXS, ?);
   verdict;

END_PROCEDURE; -- remove_unordered_bag_of_real
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of REAL. The aggregate shall be that processed by atc_add_unordered_bag_of_real.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.115 atc_is_member_bag_of_real

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *is member* for an aggregate of type BAG of REAL.

EXPRESS specification:

```
*)
PROCEDURE atc_is_member_bag_of_real(aggr : aggregate_instance);
   LOCAL
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_is_member_bag_of_real');

   purpose('Ensures that is_member returns TRUE when value '
      + 'submitted belongs to the aggregate.');
   bool := is_member(aggr, asp(1.1, ?), NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that is_member returns FALSE when value submitted '
      + 'does not belong to the aggregate.');
   bool := is_member(aggr, asp(3.3, ?), NO_ERROR, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   assert(NOT bool);

END_PROCEDURE; -- atc_is_member_bag_of_real
(*
```

Argument definitions:

aggr: (input) An aggregate of type BAG of REAL. The aggregate shall be that processed by remove_unordered_bag_of_real.

6.2.116 atc_get_current_member_bag_of_real

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get current member* for an aggregate of type BAG of REAL.

EXPRESS specification:

```
*)
PROCEDURE atc_get_current_member_bag_of_real
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
      aggr_auxiliary : BAG [0:?] OF primitive;
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_get_current_member_bag_of_real');

   purpose('Ensures that get_current_member reports an IR_NEXS error '
      + 'when iterator is not provided.');
   p := get_current_member(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that get_current_member reports an IR_NSET error '
      + 'when iterator is at the beginning.');
   beginning(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
   verdict;

   purpose('Ensures that get_current_member works correctly when '
      + 'iterator has current member set.');
   bool := next(iter, NO_ERROR, ?);
   aggr_auxiliary[1] := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr_auxiliary[2] := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr_auxiliary[3] := get_current_member(iter, NO_ERROR, ?);
   bool := macro_compare_aggregates(aggr, aggr_auxiliary);
   assert(bool);
   verdict;

   purpose('Ensures that get_current_member reports an IR_NSET error '
      + 'when iterator is at the end.');
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
   verdict;

END_PROCEDURE; -- atc_get_current_member_bag_of_real
(*
```

Argument definitions:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

aggr: (input) An aggregate of type BAG of REAL. The aggregate shall contain three values of type REAL.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.117 atc_put_current_member_bag_of_real

This abstract test case shall be used for the assessment of the implementation of the SDAI operation operation *put current member* for an aggregate of type BAG of REAL.

EXPRESS specification:

```
*)
PROCEDURE atc_put_current_member_bag_of_real
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_put_current_member_bag_of_real');

  purpose('Ensures that put_current_member reports an IR_NEXS error '
    + 'when iterator is not provided.');
  put_current_member(?, asp(3.3, ?), IR_NEXS, ?);
  verdict;

  purpose('Ensures that put_current_member reports an IR_NSET error '
    + 'when iterator is at the beginning.');
  beginning(iter, NO_ERROR, ?);
  put_current_member(iter, asp(3.3, ?), IR_NSET, iter);
  verdict;

  purpose('Ensures that put_current_member reports a VA_NSET error '
    + 'when an unset value is submitted.');
  bool := next(iter, NO_ERROR, ?);
  put_current_member(iter, ?, VA_NSET, ?);
  verdict;

  purpose('Ensures that put_current_member reports a VT_NVLD error '
    + 'when value of a wrong type is submitted.');
  put_current_member(iter, asp('something', ?), VT_NVLD, ?);
  verdict;

  purpose('Ensures that put_current_member works correctly when '
    + 'all parameters are correct.');
  p := get_current_member(iter, NO_ERROR, ?);
  put_current_member(iter, asp(3.3, ?), NO_ERROR, ?);
  bool := is_member(aggr, asp(3.3, ?), NO_ERROR, ?);
  assert(bool);
  bool := is_member(aggr, p, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that put_current_member reports an IR_NSET error '
    + 'when iterator is at the end.');
  bool := next(iter, NO_ERROR, ?);
  bool := next(iter, NO_ERROR, ?);
  put_current_member(iter, asp(3.3, ?), IR_NSET, iter);
  verdict;

END_PROCEDURE; -- atc_put_current_member_bag_of_real
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

Argument definitions:

aggr: (input) An aggregate of type BAG of REAL. Its elements shall be real numbers different than 3.3.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.118 atc_remove_current_member_bag_of_real

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *remove current member* for an aggregate of type BAG of REAL.

EXPRESS specification:

```
*)
PROCEDURE atc_remove_current_member_bag_of_real
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_remove_current_member_bag_of_real');

   purpose('Ensures that remove_current_member reports an IR_NEXS '
      + 'error when iterator is not provided.');
   bool := remove_current_member(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that remove_current_member reports an IR_NSET '
      + 'error when iterator is at the beginning.');
   beginning(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, IR_NSET, iter);
   verdict;

   purpose('Ensures that remove_current_member works correctly '
      + 'when iterator has current member set.');
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, NO_ERROR, ?);
   assert(bool);
   bool := is_member(aggr, p, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that remove_current_member returns FALSE when '
      + 'iterator refers to the last member of the aggregate.');
   bool := next(iter, NO_ERROR, ?);
   bool := remove_current_member(iter, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that remove_current_member reports an IR_NSET '
      + 'error when iterator is at the end.');
   bool := remove_current_member(iter, IR_NSET, iter);
   verdict;

END_PROCEDURE; -- atc_remove_current_member_bag_of_real
(*
```

Argument definitions:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

aggr: (input) An aggregate of type BAG of REAL. It shall contain three values of type REAL, first of which is unique.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.119 atg_array_of_integer

This abstract test group shall be used for the assessment of the implementation of the SDAI operations *get by index*, *test by index*, *put by index*, *unset value by index*, *is member*, *get current member*, *put current member*, *test current member*, and *unset value current member* for an aggregate of type ARRAY of INTEGER.

EXPRESS specification:

```
*)
PROCEDURE atg_array_of_integer
      (aggr : aggregate_instance; iter : iterator);

   (* Testing of the aggregate operation get by index. *)
   atc_get_by_index_array_of_integer(aggr);

   (* Testing of the array operation test by index. *)
   atc_test_by_index_array_of_integer(aggr);

   (* Testing of the aggregate operation put by index. *)
   atc_put_by_index_array_of_integer(aggr);

   (* Testing of the array operation unset value by index. *)
   atc_unset_value_by_index_array_of_integer(aggr);

   (* Testing of the aggregate operation is member. *)
   atc_is_member_array_of_integer(aggr);

   (* Testing of the aggregate operation put current member. *)
   atc_put_current_member_array_of_integer(iter);

   (* Testing of the aggregate operation get current member. *)
   atc_get_current_member_array_of_integer(iter);

   (* Testing of the array operation test current member. *)
   atc_test_current_member_array_of_integer(iter);

   (* Testing of the array operation unset value current member. *)
   atc_unset_value_current_member_array_of_integer(iter);

END_PROCEDURE; -- atg_array_of_integer
(*
```

Argument definitions:

aggr: (input) An aggregate of type ARRAY of INTEGER. The aggregate shall have all values unset.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.120 atc_get_by_index_array_of_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get by index* for an aggregate of type ARRAY of INTEGER.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
*)
PROCEDURE atc_get_by_index_array_of_integer
      (aggr : aggregate_instance);
   LOCAL
      p : primitive;
   END_LOCAL;

   atc('atc_get_by_index_array_of_integer');

   purpose('Ensures that get_by_index reports an IX_NVLD error when '
      + 'the index submitted is outside of the legal range.');
   p := get_by_index(aggr, -1, IX_NVLD, aggr);
   p := get_by_index(aggr, 5, IX_NVLD, aggr);
   verdict;

   purpose('Ensures that get_by_index reports a VA_NSET error when '
      + 'the array has no value at the specified position.');
   p := get_by_index(aggr, 0, VA_NSET, ?);
   verdict;

   purpose('Ensures that get_by_index works correctly when the '
      + 'index submitted is from a legal range.');
   put_by_index(aggr, 1, asp(100, ?), NO_ERROR, ?);
   p := get_by_index(aggr, 1, NO_ERROR, ?);
   assert(p = asp(100, ?));
   verdict;

END_PROCEDURE; -- atc_get_by_index_array_of_integer
(*
```

Argument definitions:

aggr: (input) An aggregate of type ARRAY of INTEGER. It shall have unset value in the first position.

6.2.121 atc_test_by_index_array_of_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *test by index* for an aggregate of type ARRAY of INTEGER.

EXPRESS specification:

```
*)
PROCEDURE atc_test_by_index_array_of_integer
      (aggr : aggregate_instance);
   LOCAL
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_test_by_index_array_of_integer');

   purpose('Ensures that test_by_index reports an IX_NVLD error when '
      + 'the index submitted is outside of the legal range.');
   bool := test_by_index(aggr, -1, IX_NVLD, aggr);
   bool := test_by_index(aggr, 5, IX_NVLD, aggr);
   verdict;

   purpose('Ensures that test_by_index returns FALSE when the array has '
      + 'no value at the specified position.');
   bool := test_by_index(aggr, 0, NO_ERROR, ?);
   assert(NOT bool);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that test_by_index returns TRUE when the array has '
      + 'some value at the specified position.');
   put_by_index(aggr, 2, asp(200, ?), NO_ERROR, ?);
   bool := test_by_index(aggr, 2, NO_ERROR, ?);
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_test_by_index_array_of_integer
(*
```

Argument definitions:

aggr: (input) An aggregate of type ARRAY of INTEGER. It shall have unset value in the first position.

6.2.122 atc_put_by_index_array_of_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *put by index* for an aggregate of type ARRAY of INTEGER.

EXPRESS specification:

```
*)
PROCEDURE atc_put_by_index_array_of_integer
       (aggr : aggregate_instance);
   LOCAL
      p : primitive;
   END_LOCAL;

   atc('atc_put_by_index_array_of_integer');

   purpose('Ensures that put_by_index reports an IX_NVLD error when '
      + 'the index submitted is outside of the legal range.');
   put_by_index(aggr, -1, asp(100, ?), IX_NVLD, aggr);
   put_by_index(aggr, 5, asp(100, ?), IX_NVLD, aggr);
   verdict;

   purpose('Ensures that put_by_index reports a VA_NSET error when '
      + 'an unset value is submitted.');
   put_by_index(aggr, 1, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_by_index reports a VT_NVLD error when value '
      + 'of a wrong type is submitted.');
   put_by_index(aggr, 1, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_by_index works correctly when all '
      + 'parameters are correct.');
   put_by_index(aggr, 3, asp(300, ?), NO_ERROR, ?);
   p := get_by_index(aggr, 3, NO_ERROR, ?);
   assert(p = asp(300, ?));
   verdict;

END_PROCEDURE; -- atc_put_by_index_array_of_integer
(*
```

Argument definitions:

aggr: (input) An aggregate of type ARRAY of INTEGER.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.123 atc_unset_value_by_index_array_of_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *unset value by index* for an aggregate of type ARRAY of INTEGER.

EXPRESS specification:

```
*)
PROCEDURE atc_unset_value_by_index_array_of_integer
      (aggr : aggregate_instance);
  LOCAL
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_unset_value_by_index_array_of_integer');

  purpose('Ensures that unset_value_by_index reports an IX_NVLD error '
    + 'when the index submitted is outside of the legal range.');
  unset_value_by_index(aggr, -1, IX_NVLD, aggr);
  unset_value_by_index(aggr, 5, IX_NVLD, aggr);
  verdict;

  purpose('Ensures that unset_value_by_index works correctly when '
    + 'the index submitted is from a legal range.');
  unset_value_by_index(aggr, 1, NO_ERROR, ?);
  verdict;

  purpose('Ensures that array value is really unset '
    + 'after unset_value_by_index operation.');
  bool := test_by_index(aggr, 1, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

END_PROCEDURE; -- atc_unset_value_by_index_array_of_integer
(*
```

Argument definitions:

aggr: (input) An aggregate of type ARRAY of INTEGER. It shall have some value in the position with index 1.

6.2.124 atc_is_member_array_of_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operations *is member* and *get member count* for an aggregate of type ARRAY of INTEGER.

EXPRESS specification:

```
*)
PROCEDURE atc_is_member_array_of_integer
      (aggr : aggregate_instance);
  LOCAL
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_is_member_array_of_integer');

  purpose('Ensures that is_member returns TRUE when value '
    + 'submitted belongs to the aggregate.');
  bool := is_member(aggr, asp(300, ?), NO_ERROR, ?);
  assert(bool);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   verdict;

   purpose('Ensures that is_member returns FALSE when value submitted '
      + 'does not belong to the aggregate.');
   bool := is_member(aggr, asp(400, ?), NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that get_member_count works correctly.');
   assert(get_member_count(aggr, NO_ERROR, ?) = 5);
   verdict;

END_PROCEDURE; -- atc_is_member_array_of_integer
(*
```

<u>Argument definitions:</u>

aggr: (input) An aggregate of type ARRAY of INTEGER. It shall contain value 300 but not 400.

6.2.125 atc_get_current_member_array_of_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *get current member* for an aggregate of type ARRAY of INTEGER.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_get_current_member_array_of_integer
      (iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_get_current_member_array_of_integer');

   purpose('Ensures that get_current_member reports an IR_NEXS error '
      + 'when iterator is not provided.');
   p := get_current_member(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that get_current_member reports an IR_NSET error '
      + 'when iterator has no current member set.');
   atEnd(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
   beginning(iter, NO_ERROR, ?);
   p := get_current_member(iter, IR_NSET, iter);
   verdict;

   purpose('Ensures that get_current_member reports a VA_NSET error '
      + 'when iterator has current member unset.');
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, VA_NSET, ?);
   verdict;

   purpose('Ensures that get_current_member works correctly when '
      + 'iterator has current member set.');
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   assert(p = asp(100, ?));
   verdict;

END_PROCEDURE; -- atc_get_current_member_array_of_integer
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type ARRAY of INTEGER. The aggregate shall be that processed by atc_put_current_member_array_of_integer.

6.2.126 atc_put_current_member_array_of_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *put current member* for an aggregate of type ARRAY of INTEGER.

EXPRESS specification:

```
*)
PROCEDURE atc_put_current_member_array_of_integer
      (iter : iterator);
   LOCAL
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_put_current_member_array_of_integer');

   purpose('Ensures that put_current_member reports an IR_NEXS error '
      + 'when iterator is not provided.');
   put_current_member(?, asp(100, ?), IR_NEXS, ?);
   verdict;

   purpose('Ensures that put_current_member reports an IR_NSET error '
      + 'when iterator has no current member set.');
   atEnd(iter, NO_ERROR, ?);
   put_current_member(iter, asp(100, ?), IR_NSET, iter);
   beginning(iter, NO_ERROR, ?);
   put_current_member(iter, asp(100, ?), IR_NSET, iter);
   verdict;

   purpose('Ensures that put_current_member reports a VA_NSET error '
      + 'when an unset value is submitted.');
   bool := next(iter, NO_ERROR, ?);
   put_current_member(iter, ?, VA_NSET, ?);
   verdict;

   purpose('Ensures that put_current_member reports a VT_NVLD error '
      + 'when value of a wrong type is submitted.');
   put_current_member(iter, asp('something', ?), VT_NVLD, ?);
   verdict;

   purpose('Ensures that put_current_member works correctly when '
      + 'parameters are correct.');
   bool := next(iter, NO_ERROR, ?);
   put_current_member(iter, asp(100, ?), NO_ERROR, ?);
   verdict;

END_PROCEDURE; -- atc_put_current_member_array_of_integer
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type ARRAY of INTEGER.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.127 atc_test_current_member_array_of_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *test current member* for an aggregate of type ARRAY of INTEGER.

EXPRESS specification:

```
*)
PROCEDURE atc_test_current_member_array_of_integer
      (iter : iterator);
  LOCAL
    int : INTEGER;
    bool : BOOLEAN;
  END_LOCAL;

  atc('atc_test_current_member_array_of_integer');

  purpose('Ensures that test_current_member reports an IR_NEXS error '
    + 'when iterator is not provided.');
  bool := test_current_member(?, IR_NEXS, ?);
  verdict;

  purpose('Ensures that test_current_member reports an IR_NSET error '
    + 'when iterator has no current member set.');
  atEnd(iter, NO_ERROR, ?);
  bool := test_current_member(iter, IR_NSET, iter);
  beginning(iter, NO_ERROR, ?);
  bool := test_current_member(iter, IR_NSET, iter);
  verdict;

  purpose('Ensures that test_current_member returns FALSE when value '
    + 'is unset at a position referred by the iterator.');
  bool := next(iter, NO_ERROR, ?);
  bool := test_current_member(iter, NO_ERROR, ?);
  assert(NOT bool);
  verdict;

  purpose('Ensures that test_current_member returns TRUE when value is '
    + 'set at a position referred by the iterator.');
  bool := next(iter, NO_ERROR, ?);
  bool := test_current_member(iter, NO_ERROR, ?);
  assert(bool);
  verdict;

END_PROCEDURE; -- atc_test_current_member_array_of_integer
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type ARRAY of INTEGER. The aggregate shall be that processed by atc_put_current_member_array_of_integer.

6.2.128 atc_unset_value_current_member_array_of_integer

This abstract test case shall be used for the assessment of the implementation of the SDAI operation *unset value current member* for an aggregate of type ARRAY of INTEGER.

EXPRESS specification:

```
*)
PROCEDURE atc_unset_value_current_member_array_of_integer
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      (iter : iterator);
   LOCAL
     bool : BOOLEAN;
   END_LOCAL;

   atc('atc_unset_value_current_member_array_of_integer');

   purpose('Ensures that unset_value_current_member reports an '
     + 'IR_NEXS error when iterator is not provided.');
   unset_value_current_member(?, IR_NEXS, ?);
   verdict;

   purpose('Ensures that unset_value_current_member reports an '
     + 'IR_NSET error when iterator has no current member set.');
   atEnd(iter, NO_ERROR, ?);
   unset_value_current_member(iter, IR_NSET, iter);
   beginning(iter, NO_ERROR, ?);
   unset_value_current_member(iter, IR_NSET, iter);
   verdict;

   purpose('Ensures that unset_value_current_member works correctly '
     + 'when iterator has current member.');
   bool := next(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   unset_value_current_member(iter, NO_ERROR, ?);
   verdict;

   purpose('Ensures that array value is really unset after '
      + 'unset_value_current_member operation.');
   bool := test_current_member(iter, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

END_PROCEDURE; -- atc_unset_value_current_member_array_of_integer
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type ARRAY of INTEGER. The aggregate shall be that processed by atc_put_current_member_array_of_integer.

6.2.129 atc_aggregate_operations_disallowed_for_list

This abstract test case shall be used for the assessment of the implementation of SDAI operations on aggregates of type LIST for the case when the SDAI operations are not allowed for LIST. In all such cases an error indicator AI_NVLD shall be reported.

EXPRESS specification:

```
*)
PROCEDURE atc_aggregate_operations_disallowed_for_list
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
     bool : BOOLEAN;
   END_LOCAL;

   atc('atc_aggregate_operations_disallowed_for_list');

   purpose('Ensures that an AI_NVLD error is reported when SET and '
     + 'BAG operations are applied to a LIST.');
   add_unordered(aggr, asp(7.7, ?), AI_NVLD, aggr);
   remove_unordered(aggr, asp(7.7, ?), AI_NVLD, aggr);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that an AI_NVLD error is reported when ARRAY '
      + 'operations are applied to a LIST.');
   bool := test_by_index(aggr, 1, AI_NVLD, aggr);
   unset_value_by_index(aggr, 1, AI_NVLD, aggr);
   bool := test_current_member(iter, AI_NVLD, aggr);
   unset_value_current_member(iter, AI_NVLD, aggr);
   verdict;

END_PROCEDURE; -- atc_aggregate_operations_disallowed_for_list
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.130 atc_aggregate_operations_disallowed_for_set

This abstract test case shall be used for the assessment of the implementation of SDAI operations on aggregates of type SET for the case when the SDAI operations are not allowed for SET. In all such cases an error indicator AI_NVLD shall be reported.

EXPRESS specification:

```
*)
PROCEDURE atc_aggregate_operations_disallowed_for_set
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_aggregate_operations_disallowed_for_set');

   purpose('Ensures that an AI_NVLD error is reported when LIST and '
      + 'ARRAY operations are applied to a SET.');
   p := get_by_index(aggr, 1, AI_NVLD, aggr);
   atEnd(iter, AI_NVLD, aggr);
   bool := previous(iter, AI_NVLD, aggr);
   put_by_index(aggr, 1, asp(UNKNOWN, ?), AI_NVLD, aggr);
   verdict;

   purpose('Ensures that an AI_NVLD error is reported when LIST '
      + 'operations are applied to a SET.');
   add_by_index(aggr, 1, asp(UNKNOWN, ?), AI_NVLD, aggr);
   remove_by_index(aggr, 1, AI_NVLD, aggr);
   add_before_current_member(iter, asp(UNKNOWN, ?), AI_NVLD, aggr);
   add_after_current_member(iter, asp(UNKNOWN, ?), AI_NVLD, aggr);
   verdict;

   purpose('Ensures that an AI_NVLD error is reported when ARRAY '
      + 'operations are applied to a SET.');
   bool := test_by_index(aggr, 1, AI_NVLD, aggr);
   unset_value_by_index(aggr, 1, AI_NVLD, aggr);
   bool := test_current_member(iter, AI_NVLD, aggr);
   unset_value_current_member(iter, AI_NVLD, aggr);
   verdict;

END_PROCEDURE; -- atc_aggregate_operations_disallowed_for_set
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

Argument definitions:

aggr: (input) An aggregate of type SET of LOGICAL.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.131 atc_aggregate_operations_disallowed_for_array

This abstract test case shall be used for the assessment of the implementation of SDAI operations on aggregates of type ARRAY for the case when the SDAI operations are not allowed for ARRAY. In all such cases an error indicator AI_NVLD shall be reported.

EXPRESS specification:

```
*)
PROCEDURE atc_aggregate_operations_disallowed_for_array
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
     bool : BOOLEAN;
   END_LOCAL;

   atc('atc_aggregate_operations_disallowed_for_array');

   purpose('Ensures that an AI_NVLD error is reported when LIST '
     + 'operations are applied to an ARRAY.');
   add_by_index(aggr, 1, asp(10, ?), AI_NVLD, aggr);
   remove_by_index(aggr, 1, AI_NVLD, aggr);
   add_before_current_member(iter, asp(10, ?), AI_NVLD, aggr);
   add_after_current_member(iter, asp(10, ?), AI_NVLD, aggr);
   verdict;

   purpose('Ensures that an AI_NVLD error is reported when SET and '
     + 'BAG operations are applied to an ARRAY.');
   add_unordered(aggr, asp(10, ?), AI_NVLD, aggr);
   remove_unordered(aggr, asp(10, ?), AI_NVLD, aggr);
   verdict;

   purpose('Ensures that an AI_NVLD error is reported when LIST, SET '
     + 'and BAG operation remove_current_member is applied to an ARRAY.');
   bool := remove_current_member(iter, AI_NVLD, aggr);
   verdict;

END_PROCEDURE; -- atc_aggregate_operations_disallowed_for_array
(*
```

Argument definitions:

aggr: (input) An aggregate of type ARRAY of INTEGER.
iter: (input) An iterator over an aggregate specified by the first argument.

6.2.132 atg_nested_aggregate

This abstract test group shall be used for the assessment of the implementation of SDAI operations on nested aggregates.

EXPRESS specification:

```
*)
PROCEDURE atg_nested_aggregate(modl : sdai_model);
   LOCAL
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
      inst : application_instance := create_entity_instance('ESTS.EPSILON',
        modl, NO_ERROR, ?);
      aggr : aggregate_instance;
      iter : iterator;
      aggr_created : aggregate_primitive; -- LIST [1:3] OF REAL
   END_LOCAL;

   (* An aggregate of type LIST of select type nu in ESTS schema and its
      iterator are created. *)
   aggr := create_aggregate_instance(inst, 'ESTS.EPSILON.E2', ?,
      NO_ERROR, ?);
   iter := create_iterator(aggr, NO_ERROR, ?);

   purpose('Ensures that an AI_NVLD error is reported when SET and '
      + 'BAG operation create aggregate instance unordered is applied '
      + 'to a LIST.');
   aggr_created := create_aggregate_instance_unordered(aggr, ['ESTS.CHI'],
      AI_NVLD, aggr);
   verdict;

   (* Testing the SDAI operations on aggregate creation for an aggregate of
      type LIST of LIST. *)
   atg_list_of_list(aggr, iter);

   (* An aggregate of type SET of select type rho in ESTS schema and its
      iterator are created. *)
   aggr := create_aggregate_instance(inst, 'ESTS.EPSILON.E6', ?,
      NO_ERROR, ?);
   iter := create_iterator(aggr, NO_ERROR, ?);

   (* Checks cases when aggregate creation operations are illegal
      for the SET. *)
   atc_aggregate_creation_operations_disallowed_for_set(aggr, iter);

   (* Testing the SDAI operations on aggregate creation for an aggregate
      of type SET of SET. *)
   atg_set_of_set(aggr, iter);

   (* An aggregate of type ARRAY of select type nu in ESTS schema and its
      iterator are created. *)
   aggr := create_aggregate_instance(inst, 'ESTS.EPSILON.E4', ?,
      NO_ERROR, ?);
   iter := create_iterator(aggr, NO_ERROR, ?);

   (* Checks cases when aggregate creation operations are illegal
      for the ARRAY. *)
   atc_aggregate_creation_operations_disallowed_for_array(aggr, iter);

   (* Testing the SDAI operations on aggregate creation for an aggregate
      of type ARRAY of LIST. *)
   atg_array_of_list(aggr, iter, modl);

END_PROCEDURE; -- atg_nested_aggregate
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Argument definitions:

modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.133 atg_list_of_list

This abstract test group shall be used for the assessment of the implementation of SDAI operations *add aggregate instance by index, create aggregate instance by index, create aggregate instance before*

© ISO 2003 — All rights reserved

current member, *create aggregate instance after current member*, and *create aggregate instance as current member* for an aggregate of type LIST of select data type, where the set of selections in the select data type includes LIST of REAL.

EXPRESS specification:

```
*)
PROCEDURE atg_list_of_list(aggr : aggregate_instance; iter : iterator);

  (* Testing of the list operation add aggregate instance by index. *)
  atc_add_aggregate_instance_by_index_list_of_list(aggr);

  (* Testing of the aggregate operation create aggregate instance
     by index. *)
  atc_create_aggregate_instance_by_index_list_of_list(aggr);

  (* Making the given aggregate empty. *)
  macro_clear_aggregate(aggr);

  (* Testing of the list operation create aggregate instance before
     current member. *)
  atc_create_aggregate_instance_before_current_member_list_of_list(iter);

  (* Making the given aggregate empty. *)
  macro_clear_aggregate(aggr);

  (* Testing of the list operation create aggregate instance after
     current member. *)
  atc_create_aggregate_instance_after_current_member_list_of_list(iter);

  (* Testing of the aggregate operation create aggregate instance as
     current member. *)
  atc_create_aggregate_instance_as_current_member_list_of_list(iter);

END_PROCEDURE; -- atg_list_of_list
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of select data type nu in ESTS schema. The aggregate shall be empty.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.134 atc_add_aggregate_instance_by_index_list_of_list

This abstract test case shall be used for the assessment of the implementation of SDAI operation *add aggregate instance by index* for an aggregate of type LIST of select data type, where the set of selections in the select data type includes LIST of REAL.

EXPRESS specification:

```
*)
PROCEDURE atc_add_aggregate_instance_by_index_list_of_list
      (aggr : aggregate_instance);
  LOCAL
    aggr_added : aggregate_primitive; -- LIST [1:3] OF REAL
    aggr_returned : aggregate_primitive; -- LIST [1:3] OF REAL
    count : INTEGER;
    bool : BOOLEAN;
    p : primitive;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   END_LOCAL;

   atc('atc_add_aggregate_instance_by_index_list_of_list');

   purpose('Ensures that indexing in the LIST starts from 1.');
   aggr_added := add_aggregate_instance_by_index(aggr, 0, ['ESTS.CHI'],
     IX_NVLD, aggr);
   verdict;

   (* Making an aggregate initially nonempty. *)
   add_by_index(aggr, 1, asp(10, ['ESTS.XI']), NO_ERROR, ?);
   add_by_index(aggr, 2, asp(20, ['ESTS.XI']), NO_ERROR, ?);
   add_by_index(aggr, 3, asp(30, ['ESTS.XI']), NO_ERROR, ?);

   purpose('Ensures that add_aggregate_instance_by_index reports a '
     + 'VT_NVLD error when list of defined types is either empty '
     + '(but should not be such) or contains wrong elements or '
     + 'is not submitted at all though is needed.');
   aggr_added := add_aggregate_instance_by_index(aggr, 3, [], VT_NVLD, ?);
   aggr_added := add_aggregate_instance_by_index(aggr, 3, ['ESTS.XI'],
     VT_NVLD, ?);
   aggr_added := add_aggregate_instance_by_index(aggr, 3, ?, VT_NVLD, ?);
   verdict;

   purpose('Ensures that add_aggregate_instance_by_index works correctly '
     + 'when all parameters are correct.');
   aggr_added := add_aggregate_instance_by_index(aggr, 3, ['ESTS.CHI'],
     NO_ERROR, ?);
   verdict;

   purpose('Ensures that an aggregate created is empty.');
   count := get_member_count(aggr_added, NO_ERROR, ?);
   assert(count = 0);
   verdict;

   purpose('Ensures that an aggregate of a required type is created and '
     + 'is added in the correct place.');
   add_by_index(aggr_added, 1, asp(5.5, ?), NO_ERROR, ?);
   p := get_by_index(aggr, 3, NO_ERROR, ?);
   aggr_returned := macro_convert_primitive_to_aggregate(p);
   bool := is_member(aggr_returned, asp(5.5, ?), NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that add_aggregate_instance_by_index reports '
     + 'an IX_NVLD error when the index submitted exceeds the count '
     + 'of aggregate members plus one.');
   aggr_added := add_aggregate_instance_by_index(aggr, 6, ['ESTS.CHI'],
     IX_NVLD, aggr);
   verdict;

END_PROCEDURE; -- atc_add_aggregate_instance_by_index_list_of_list
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Argument definitions:

aggr: (input) An aggregate of type LIST of select data type. The aggregate shall be empty.

6.2.135 atc_create_aggregate_instance_by_index_list_of_list

This abstract test case shall be used for the assessment of the implementation of SDAI operation *create aggregate instance by index* for an aggregate of type LIST of select data type, where the set of selections in the select data type includes LIST of REAL.

© ISO 2003 — All rights reserved

EXPRESS specification:

```
*)
PROCEDURE atc_create_aggregate_instance_by_index_list_of_list
      (aggr : aggregate_instance);
  LOCAL
    aggr_created : aggregate_primitive; -- LIST [1:3] OF REAL
    aggr_returned : aggregate_primitive; -- LIST [1:3] OF REAL
    count : INTEGER;
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_create_aggregate_instance_by_index_list_of_list');

  purpose('Ensures that create_aggregate_instance_by_index reports an '
    + 'IX_NVLD error when the index submitted is outside of the '
    + 'legal range.');
  aggr_created := create_aggregate_instance_by_index(aggr, 0,
    ['ESTS.CHI'], IX_NVLD, aggr);
  aggr_created := create_aggregate_instance_by_index(aggr, 5,
    ['ESTS.CHI'], IX_NVLD, aggr);
  verdict;

  purpose('Ensures that create_aggregate_instance_by_index reports a '
    + 'VT_NVLD error when list of defined types is either empty '
    + '(but should not be such) or contains wrong elements or is '
    + 'not submitted at all though is needed.');
  aggr_created := create_aggregate_instance_by_index(aggr, 1, [],
    VT_NVLD, ?);
  aggr_created := create_aggregate_instance_by_index(aggr, 1,
    ['ESTS.XI'], VT_NVLD, ?);
  aggr_created := create_aggregate_instance_by_index(aggr, 1, ?,
    VT_NVLD, ?);
  verdict;

  purpose('Ensures that create_aggregate_instance_by_index works '
    + 'correctly when all parameters are correct.');
  aggr_created := create_aggregate_instance_by_index(aggr, 1,
    ['ESTS.CHI'], NO_ERROR, ?);
  verdict;

  purpose('Ensures that an aggregate created is empty.');
  count := get_member_count(aggr_created, NO_ERROR, ?);
  assert(count = 0);
  verdict;

  purpose('Ensures that an aggregate of a required type is created and '
    + 'is assigned to the correct place.');
  add_by_index(aggr_created, 1, asp(99.99, ?), NO_ERROR, ?);
  p := get_by_index(aggr, 1, NO_ERROR, ?);
  aggr_returned := macro_convert_primitive_to_aggregate(p);
  bool := is_member(aggr_returned, asp(99.99, ?), NO_ERROR, ?);
  assert(bool);
  verdict;

END_PROCEDURE; -- atc_create_aggregate_instance_by_index_list_of_list
(*
```

Argument definitions:

aggr: (input) An aggregate of type LIST of select data type. The aggregate shall be that processed by atc_add_aggregate_instance_by_index_list_of_list.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

6.2.136 atc_create_aggregate_instance_before_current_member_list_of_list

This abstract test case shall be used for the assessment of the implementation of SDAI operation *create aggregate instance before current member* for an aggregate of type LIST of select data type, where the set of selections in the select data type includes LIST of REAL.

EXPRESS specification:

```
*)
PROCEDURE atc_create_aggregate_instance_before_current_member_list_of_list
      (iter : iterator);
   LOCAL
      aggr_added : aggregate_primitive; -- LIST [1:3] OF REAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_create_aggregate_instance_before_current_member_list_of_list');

   purpose('Ensures that create_aggregate_instance_before_current_member '
      + 'reports an IR_NEXS error when iterator is not provided.');
   aggr_added := create_aggregate_instance_before_current_member(?,
      ['ESTS.CHI'], IR_NEXS, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance_before_current_member '
      + 'reports a VT_NVLD error when list of defined types is either '
      + 'empty (but should not be such) or contains wrong elements or is '
      + 'not submitted at all though is needed.');
   beginning(iter, NO_ERROR, ?);
   aggr_added := create_aggregate_instance_before_current_member(iter, [],
      VT_NVLD, ?);
   aggr_added := create_aggregate_instance_before_current_member(iter,
      ['ESTS.XI'], VT_NVLD, ?);
   aggr_added := create_aggregate_instance_before_current_member(iter, ?,
      VT_NVLD, ?);
   verdict;

   purpose('Ensures that after '
      + 'create_aggregate_instance_before_current_member '
      + 'operation for an empty list iterator is positioned at the end '
      + 'of the list (there is no current member).');
   aggr_added := create_aggregate_instance_before_current_member(iter,
      ['ESTS.CHI'], NO_ERROR, ?);
   add_by_index(aggr_added, 1, asp(2.2, ?), NO_ERROR, ?);
   bool := previous(iter, NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that create_aggregate_instance_before_current_member '
      + 'works correctly when all parameters are correct.');
   aggr_added := create_aggregate_instance_before_current_member(iter,
      ['ESTS.CHI'], NO_ERROR, ?);
   add_by_index(aggr_added, 1, asp(1.1, ?), NO_ERROR, ?);
   atEnd(iter, NO_ERROR, ?);
   aggr_added := create_aggregate_instance_before_current_member(iter,
      ['ESTS.CHI'], NO_ERROR, ?);
   add_by_index(aggr_added, 1, asp(3.3, ?), NO_ERROR, ?);
   verdict;

   purpose('Checking if created aggregates were placed in '
      + 'correct positions.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   beginning(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   aggr_added := macro_convert_primitive_to_aggregate(p);
   p := get_by_index(aggr_added, 1, NO_ERROR, ?);
   assert(p = asp(1.1, ?));
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   aggr_added := macro_convert_primitive_to_aggregate(p);
   p := get_by_index(aggr_added, 1, NO_ERROR, ?);
   assert(p = asp(2.2, ?));
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   aggr_added := macro_convert_primitive_to_aggregate(p);
   p := get_by_index(aggr_added, 1, NO_ERROR, ?);
   assert(p = asp(3.3, ?));
   verdict;

END_PROCEDURE;
--atc_create_aggregate_instance_before_current_member_list_of_list
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of select data type. The aggregate shall be empty.

6.2.137 atc_create_aggregate_instance_after_current_member_list_of_list

This abstract test case shall be used for the assessment of the implementation of SDAI operation *create aggregate instance after current member* for an aggregate of type LIST of select data type, where the set of selections in the select data type includes LIST of REAL.

EXPRESS specification:

```
*)
PROCEDURE atc_create_aggregate_instance_after_current_member_list_of_list
      (iter : iterator);
   LOCAL
      aggr_added : aggregate_primitive; -- LIST [1:3] OF REAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_create_aggregate_instance_after_current_member_list_of_list');

   purpose('Ensures that create_aggregate_instance_after_current_member '
      + 'reports an IR_NEXS error when iterator is not provided.');
   aggr_added := create_aggregate_instance_after_current_member(?,
      ['ESTS.CHI'], IR_NEXS, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance_after_current_member '
      + 'reports a VT_NVLD error when list of defined types is either '
      + 'empty (but should not be such) or contains wrong elements or is '
      + 'not submitted at all though is needed.');
   beginning(iter, NO_ERROR, ?);
   aggr_added := create_aggregate_instance_after_current_member(iter, [],
      VT_NVLD, ?);
   aggr_added := create_aggregate_instance_after_current_member(iter,
      ['ESTS.XI'], VT_NVLD, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   aggr_added := create_aggregate_instance_after_current_member(iter, ?,
     VT_NVLD, ?);
   verdict;

   purpose('Ensures that after '
     + 'create_aggregate_instance_after_current_member '
     + 'operation for an empty list iterator is positioned at the '
     + 'beginning of the list (there is no current member).');
   aggr_added := create_aggregate_instance_after_current_member(iter,
     ['ESTS.CHI'], NO_ERROR, ?);
   add_by_index(aggr_added, 1, asp(2.2, ?), NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   assert(bool);
   verdict;

   purpose('Ensures that create_aggregate_instance_after_current_member '
     + 'works correctly when all parameters are correct.');
   aggr_added := create_aggregate_instance_after_current_member(iter,
     ['ESTS.CHI'], NO_ERROR, ?);
   add_by_index(aggr_added, 1, asp(3.3, ?), NO_ERROR, ?);
   beginning(iter, NO_ERROR, ?);
   aggr_added := create_aggregate_instance_after_current_member(iter,
     ['ESTS.CHI'], NO_ERROR, ?);
   add_by_index(aggr_added, 1, asp(1.1, ?), NO_ERROR, ?);
   verdict;

   purpose('Checking if created aggregates were placed in '
     + 'correct positions.');
   beginning(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   aggr_added := macro_convert_primitive_to_aggregate(p);
   p := get_by_index(aggr_added, 1, NO_ERROR, ?);
   assert(p = asp(1.1, ?));
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   aggr_added := macro_convert_primitive_to_aggregate(p);
   p := get_by_index(aggr_added, 1, NO_ERROR, ?);
   assert(p = asp(2.2, ?));
   bool := next(iter, NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   aggr_added := macro_convert_primitive_to_aggregate(p);
   p := get_by_index(aggr_added, 1, NO_ERROR, ?);
   assert(p = asp(3.3, ?));
   verdict;

END_PROCEDURE;
-- atc_create_aggregate_instance_after_current_member_list_of_list
(*
```

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of select data type. The aggregate shall be empty.

6.2.138 atc_create_aggregate_instance_as_current_member_list_of_list

This abstract test case shall be used for the assessment of the implementation of SDAI operation *create aggregate instance as current member* for an aggregate of type LIST of select data type, where the set of selections in the select data type includes LIST of REAL.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
*)
PROCEDURE atc_create_aggregate_instance_as_current_member_list_of_list
      (iter : iterator);
   LOCAL
      aggr_created : aggregate_primitive; -- LIST [1:3] OF REAL
      aggr_returned : aggregate_primitive; -- LIST [1:3] OF REAL
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_create_aggregate_instance_as_current_member_list_of_list');

   purpose('Ensures that create_aggregate_instance_as_current_member '
      + 'reports an IR_NEXS error when iterator is not provided.');
   aggr_created := create_aggregate_instance_as_current_member(?,
      ['ESTS.CHI'], IR_NEXS, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance_as_current_member '
      + 'reports an IR_NSET error when iterator has no current '
      + 'member set.');
   atEnd(iter, NO_ERROR, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter,
      ['ESTS.CHI'], IR_NSET, iter);
   beginning(iter, NO_ERROR, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter,
      ['ESTS.CHI'], IR_NSET, iter);
   verdict;

   purpose('Ensures that create_aggregate_instance_as_current_member '
      + 'reports a VT_NVLD error when list of defined types is either '
      + 'empty (but should not be such) or contains wrong elements or '
      + 'is not submitted at all though is needed.');
   bool := next(iter, NO_ERROR, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter, [],
      VT_NVLD, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter,
      ['ESTS.XI'], VT_NVLD, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter, ?,
      VT_NVLD, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance_as_current_member '
      + 'works correctly when all parameters are correct.');
   aggr_created := create_aggregate_instance_as_current_member(iter,
      ['ESTS.CHI'], NO_ERROR, ?);
   verdict;

   purpose('Ensures that an aggregate created is empty.');
   assert(get_member_count(aggr_created, NO_ERROR, ?) = 0);
   verdict;

   purpose('Ensures that an aggregate of a required type is created and '
      + 'is assigned to the correct place.');
   add_by_index(aggr_created, 1, asp(7.7, ?), NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   aggr_returned := macro_convert_primitive_to_aggregate(p);
   p := get_by_index(aggr_returned, 1, NO_ERROR, ?);
   assert(p = asp(7.7, ?));
   verdict;

END_PROCEDURE;
-- atc_create_aggregate_instance_as_current_member_list_of_list
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Argument definitions:

iter: (input) An iterator over an aggregate of type LIST of select data type. The aggregate shall be that processed by atc_create_aggregate_instance_after_current_member_list_of_list.

6.2.139 atg_set_of_set

This abstract test group shall be used for the assessment of the implementation of SDAI operations *create aggregate instance unordered* and *create aggregate instance as current member* for an aggregate of type SET of select data type, where the set of selections in the select data type includes SET of data type nu in ESTS schema.

EXPRESS specification:

```
*)
PROCEDURE atg_set_of_set(aggr : aggregate_instance; iter : iterator);

   (* Testing of the aggregate operation create aggregate instance
      unordered. *)
   atc_create_aggregate_instance_unordered_set_of_set(aggr, iter);

   (* Testing of the aggregate operation create aggregate instance as
      current member. *)
   atc_create_aggregate_instance_as_current_member_set_of_set(aggr, iter);

END_PROCEDURE; -- atg_set_of_set
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of select data type rho in ESTS schema. The aggregate shall be empty.
iter: (input) An iterator over an aggregate specified by the first argument.

6.2.140 atc_create_aggregate_instance_unordered_set_of_set

This abstract test case shall be used for the assessment of the implementation of SDAI operation *create aggregate instance unordered* for an aggregate of type SET of select data type, where the set of selections in the select data type includes SET of data type nu in ESTS schema.

EXPRESS specification:

```
*)
PROCEDURE atc_create_aggregate_instance_unordered_set_of_set
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
      aggr_created : aggregate_primitive; -- SET [0:3] OF nu
      aggr_auxiliary : SET [1:?] OF primitive;
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_create_aggregate_instance_unordered_set_of_set');

   purpose('Ensures that create_aggregate_instance_unordered reports a '
      + 'VT_NVLD error when list of defined types is either empty '
      + '(but should not be such) or contains wrong elements or '
      + 'is not submitted at all though is needed.');
   aggr_created := create_aggregate_instance_unordered(aggr, [],
      VT_NVLD, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   aggr_created := create_aggregate_instance_unordered(aggr,
      ['ESTS.XI'], VT_NVLD, ?);
   aggr_created := create_aggregate_instance_unordered(aggr, ?,
      VT_NVLD, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance_unordered works '
      + 'correctly when all parameters are correct.');
   aggr_created := create_aggregate_instance_unordered(aggr,
      ['ESTS.UPSILON'], NO_ERROR, ?);
   aggr_created := create_aggregate_instance_unordered(aggr,
      ['ESTS.UPSILON'], NO_ERROR, ?);
   aggr_created := create_aggregate_instance_unordered(aggr,
      ['ESTS.UPSILON'], NO_ERROR, ?);
   verdict;

   purpose('Ensures that an aggregate created is empty.');
   assert(get_member_count(aggr_created, NO_ERROR, ?) = 0);
   verdict;

   purpose('Ensures that an aggregate of a required type is created.');
   add_unordered(aggr_created, asp(10, ['ESTS.XI']), NO_ERROR, ?);
   verdict;

   purpose('Ensures that all created aggregates are added to the top '
      + 'level aggregate.');
   beginning(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr_auxiliary[1] := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr_auxiliary[2] := get_current_member(iter, NO_ERROR, ?);
   bool := next(iter, NO_ERROR, ?);
   aggr_auxiliary[3] := get_current_member(iter, NO_ERROR, ?);
   bool := macro_compare_aggregates(aggr, aggr_auxiliary);
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_create_aggregate_instance_unordered_set_of_set
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of select data type. The aggregate shall be empty.

6.2.141 atc_create_aggregate_instance_as_current_member_set_of_set

This abstract test case shall be used for the assessment of the implementation of SDAI operation *create aggregate instance as current member* for an aggregate of type SET of select data type, where the set of selections in the select data type includes SET of data type nu in ESTS schema.

EXPRESS specification:

```
*)
PROCEDURE atc_create_aggregate_instance_as_current_member_set_of_set
     (aggr : aggregate_instance; iter : iterator);
   LOCAL
      aggr_created : aggregate_primitive; -- SET [0:3] OF nu
      bool : BOOLEAN;
      p : primitive;
   END_LOCAL;

   atc('atc_create_aggregate_instance_as_current_member_set_of_set');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   purpose('Ensures that create_aggregate_instance_as_current_member '
      + 'reports an IR_NEXS error when iterator is not provided.');
   aggr_created := create_aggregate_instance_as_current_member(?,
      ['ESTS.UPSILON'], IR_NEXS, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance_as_current_member '
      + 'reports an IR_NSET error when iterator has no current '
      + 'member set.');
   beginning(iter, NO_ERROR, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter,
      ['ESTS.UPSILON'], IR_NSET, iter);
   verdict;

   purpose('Ensures that create_aggregate_instance_as_current_member '
      + 'reports a VT_NVLD error when list of defined types is either '
      + 'empty (but should not be such) or contains wrong elements or '
      + 'is not submitted at all though is needed.');
   bool := next(iter, NO_ERROR, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter, [],
      VT_NVLD, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter,
      ['ESTS.XI'], VT_NVLD, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter, ?,
      VT_NVLD, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance_as_current_member '
      + 'works correctly when all parameters are correct.');
   p := get_current_member(iter, NO_ERROR, ?);
   aggr_created := create_aggregate_instance_as_current_member(iter,
      ['ESTS.UPSILON'], NO_ERROR, ?);
   bool := is_member(aggr, aggr_created, NO_ERROR, ?);
   assert(bool);
   bool := is_member(aggr, p, NO_ERROR, ?);
   assert(NOT bool);
   verdict;

   purpose('Ensures that an aggregate created is empty.');
   assert(get_member_count(aggr_created, NO_ERROR, ?) = 0);
   verdict;

   purpose('Ensures that an aggregate of a required type is created.');
   add_unordered(aggr_created, asp(10, ['ESTS.XI']), NO_ERROR, ?);
   verdict;

END_PROCEDURE;
-- atc_create_aggregate_instance_as_current_member_set_of_set
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET of select data type. The aggregate shall be that processed by atc_create_aggregate_instance_unordered_set_of_set.

iter: (input) An iterator over an aggregate specified by the first argument.

6.2.142 atc_aggregate_creation_operations_disallowed_for_set

This abstract test case shall be used for the assessment of the implementation of SDAI operation for the creation of nested aggregates where the parent aggregate is of type SET and this is disallowed for that operation. In all such cases an error indicator AI_NVLD shall be reported.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

EXPRESS specification:

```
*)
PROCEDURE atc_aggregate_creation_operations_disallowed_for_set
      (aggr : aggregate_instance; iter : iterator);
  LOCAL
    aggr_created : aggregate_primitive; -- SET [0:3] OF nu
  END_LOCAL;

  atc('atc_aggregate_creation_operations_disallowed_for_set');

  purpose('Ensures that an AI_NVLD error is reported when LIST and ARRAY '
    + 'operation create aggregate instance by index is applied to '
    + 'a SET.');
  aggr_created := create_aggregate_instance_by_index(aggr, 1,
    ['ESTS.UPSILON'], AI_NVLD, aggr);
  verdict;

  purpose('Ensures that an AI_NVLD error is reported when LIST '
    + 'operations are applied to a SET.');
  aggr_created := add_aggregate_instance_by_index(aggr, 1,
    ['ESTS.UPSILON'], AI_NVLD, aggr);
  aggr_created := create_aggregate_instance_before_current_member(iter,
    ['ESTS.UPSILON'], AI_NVLD, aggr);
  aggr_created := create_aggregate_instance_after_current_member(iter,
    ['ESTS.UPSILON'], AI_NVLD, aggr);
  verdict;

END_PROCEDURE; -- atc_aggregate_creation_operations_disallowed_for_set
(*
```

Argument definitions:

aggr: (input) An aggregate of type SET.
iter: (input) An iterator over an aggregate specified by the first argument.

6.2.143 atg_array_of_list

This abstract test group shall be used for the assessment of the implementation of SDAI operations *create aggregate instance by index* and *create aggregate instance as current member* for an aggregate of type ARRAY of select data type, where the set of selections in the select data type includes LIST of entity data type.

EXPRESS specification:

```
*)
PROCEDURE atg_array_of_list
      (aggr : aggregate_instance; iter : iterator; modl : sdai_model);
  LOCAL
    inst : application_instance := create_entity_instance('ESTS.OMEGA',
      modl, NO_ERROR, ?);
  END_LOCAL;

  (* Testing of the aggregate operation create aggregate instance by
     index. *)
  atc_create_aggregate_instance_by_index_array_of_list(aggr, inst);

  (* Testing of the aggregate operation create aggregate instance as
     current member. *)
  atc_create_aggregate_instance_as_current_member_array_of_list(aggr,
    iter, inst);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
END_PROCEDURE; -- atg_array_of_list
(*
```

Argument definitions:

aggr: (input) An aggregate of type ARRAY of select data type omicron (what is tantamount to nu) in ESTS schema.

iter: (input) An iterator over an aggregate specified by the first argument.
modl: (input) An SDAI-model in read-write mode, based on the ESTS schema.

6.2.144 atc_create_aggregate_instance_by_index_array_of_list

This abstract test case shall be used for the assessment of the implementation of SDAI operation *create aggregate instance by index* for an aggregate of type ARRAY of select data type, where the set of selections in the select data type includes LIST of entity data type.

EXPRESS specification:

```
*)
PROCEDURE atc_create_aggregate_instance_by_index_array_of_list
      (aggr : aggregate_instance; inst : application_instance);
   LOCAL
     aggr_created : aggregate_primitive; -- LIST [1:?] OF omega
     aggr_returned : aggregate_primitive; -- LIST [1:?] OF omega
     bool : BOOLEAN;
     p : primitive;
   END_LOCAL;

   atc('atc_create_aggregate_instance_by_index_array_of_list');

   purpose('Ensures that create_aggregate_instance_by_index reports '
     + 'an IX_NVLD error when the index submitted is outside of the '
     + 'legal range.');
   aggr_created := create_aggregate_instance_by_index(aggr, 0,
     ['ESTS.PHI'], IX_NVLD, aggr);
   aggr_created := create_aggregate_instance_by_index(aggr, 4,
     ['ESTS.PHI'], IX_NVLD, aggr);
   verdict;

   purpose('Ensures that create_aggregate_instance_by_index reports '
     + 'a VT_NVLD error when list of defined types is either empty '
     + '(but should not be such) or contains wrong elements or '
     + 'is not submitted at all though is needed.');
   aggr_created := create_aggregate_instance_by_index(aggr, 1, [],
     VT_NVLD, ?);
   aggr_created := create_aggregate_instance_by_index(aggr, 1,
     ['ESTS.XI'], VT_NVLD, ?);
   aggr_created := create_aggregate_instance_by_index(aggr, 1, ?,
     VT_NVLD, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance_by_index works '
     + 'correctly when all parameters are correct.');
   aggr_created := create_aggregate_instance_by_index(aggr, 1,
     ['ESTS.PHI'], NO_ERROR, ?);
   verdict;

   purpose('Ensures that an aggregate created is empty.');
   assert(get_member_count(aggr_created, NO_ERROR, ?) = 0);
   verdict;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
   purpose('Ensures that an aggregate of a required type is created and '
     + 'is assigned to the correct place.');
   add_by_index(aggr_created, 1, asp(inst, ?), NO_ERROR, ?);
   p := get_by_index(aggr, 1, NO_ERROR, ?);
   aggr_returned := macro_convert_primitive_to_aggregate(p);
   bool := is_member(aggr_returned, asp(inst, ?), NO_ERROR, ?);
   assert(bool);
   verdict;

END_PROCEDURE; -- atc_create_aggregate_instance_by_index_array_of_list
(*
```

<u>Argument definitions:</u>

aggr: (input) An aggregate of type ARRAY of select data type.
inst: (input) An instance of entity type omega in ESTS schema.

6.2.145 atc_create_aggregate_instance_as_current_member_array_of_list

This abstract test case shall be used for the assessment of the implementation of SDAI operation *create aggregate instance as current member* for an aggregate of type ARRAY of select data type, where the set of selections in the select data type includes LIST of entity data type.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_create_aggregate_instance_as_current_member_array_of_list
     (iter : iterator; inst : application_instance);
  LOCAL
    aggr_created : aggregate_primitive; -- LIST [1:?] OF omega
    aggr_returned : aggregate_primitive; -- LIST [1:?] OF omega
    bool : BOOLEAN;
    p : primitive;
  END_LOCAL;

  atc('atc_create_aggregate_instance_as_current_member_array_of_list');

  purpose('Ensures that create_aggregate_instance_as_current_member '
    + 'reports an IR_NEXS error when iterator is not provided.');
  aggr_created := create_aggregate_instance_as_current_member(?,
    ['ESTS.PHI'], IR_NEXS, ?);
  verdict;

  purpose('Ensures that create_aggregate_instance_as_current_member '
    + 'reports an IR_NSET error when iterator has no current '
    + 'member set.');
  atEnd(iter, NO_ERROR, ?);
  aggr_created := create_aggregate_instance_as_current_member(iter,
    ['ESTS.PHI'], IR_NSET, iter);
  beginning(iter, NO_ERROR, ?);
  aggr_created := create_aggregate_instance_as_current_member(iter,
    ['ESTS.PHI'], IR_NSET, iter);
  verdict;

  purpose('Ensures that create_aggregate_instance_as_current_member '
    + 'reports a VT_NVLD error when list of defined types is either '
    + 'empty (but should not be such) or contains wrong elements or '
    + 'is not submitted at all though is needed.');
  bool := next(iter, NO_ERROR, ?);
  aggr_created := create_aggregate_instance_as_current_member(iter, [],
    VT_NVLD, ?);
  aggr_created := create_aggregate_instance_as_current_member(iter,
    ['ESTS.XI'], VT_NVLD, ?);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   aggr_created := create_aggregate_instance_as_current_member(iter, ?,
      VT_NVLD, ?);
   verdict;

   purpose('Ensures that create_aggregate_instance_as_current_member '
      + 'works correctly when all parameters are correct.');
   aggr_created := create_aggregate_instance_as_current_member(iter,
      ['ESTS.PHI'], NO_ERROR, ?);
   verdict;

   purpose('Ensures that an aggregate created is empty.');
   assert(get_member_count(aggr_created, NO_ERROR, ?) = 0);
   verdict;

   purpose('Ensures that an aggregate of a required type is created and '
      + 'is assigned to the correct place.');
   add_by_index(aggr_created, 1, asp(inst, ?), NO_ERROR, ?);
   p := get_current_member(iter, NO_ERROR, ?);
   aggr_returned := macro_convert_primitive_to_aggregate(p);
   bool := is_member(aggr_returned, asp(inst, ?), NO_ERROR, ?);
   assert(bool);
   verdict;

END_PROCEDURE;
-- atc_create_aggregate_instance_as_current_member_array_of_list
(*
```

<u>Argument definitions:</u>

iter: (input) An iterator over an aggregate of type ARRAY of select data type.
inst: (input) An instance of entity type omega in ESTS schema.

6.2.146 atc_aggregate_creation_operations_disallowed_for_array

This abstract test case shall be used for the assessment of the implementation of SDAI operation for the creation of nested aggregates where the parent aggregate is of type ARRAY and this is disallowed for that operation. In all such cases an error indicator AI_NVLD shall be reported.

<u>EXPRESS specification:</u>

```
*)
PROCEDURE atc_aggregate_creation_operations_disallowed_for_array
      (aggr : aggregate_instance; iter : iterator);
   LOCAL
      aggr_created : aggregate_primitive; -- LIST [1:?] OF omega
      bool : BOOLEAN;
   END_LOCAL;

   atc('atc_aggregate_creation_operations_disallowed_for_array');

   purpose('Ensures that an AI_NVLD error is reported when LIST '
      + 'operations are applied to an ARRAY.');
   aggr_created := add_aggregate_instance_by_index(aggr, 1, ['ESTS.PHI'],
      AI_NVLD, aggr);
   aggr_created := create_aggregate_instance_before_current_member(iter,
      ['ESTS.PHI'], AI_NVLD, aggr);
   aggr_created := create_aggregate_instance_after_current_member(iter,
      ['ESTS.PHI'], AI_NVLD, aggr);
   verdict;

   purpose('Ensures that an AI_NVLD error is reported when SET and '
      + 'BAG operation create aggregate instance unordered is applied '
      + 'to an ARRAY.');
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
  aggr_created := create_aggregate_instance_unordered(aggr, ['ESTS.PHI'],
    AI_NVLD, aggr);
  verdict;

END_PROCEDURE; -- atc_aggregate_creation_operations_disallowed_for_array
(*
```

Argument definitions:

aggr: (input) An aggregate of type ARRAY.
iter: (input) An iterator over an aggregate specified by the first argument.

6.2.147 macro_get_closed_repository

This macro function shall return a repository from the known_servers but not active_servers set of the session. If closed, a session shall be opened. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_get_closed_repository : sdai_repository;
  ;
END_FUNCTION; -- macro_get_closed_repository
(*
```

Argument definitions:

result: (output) A closed repository.

6.2.148 macro_get_open_repository

This macro function shall return a repository from the active_servers set of the session. If closed, a session shall be opened. Also, if level 3 of transactions is supported, then transaction read-write access shall be started. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_get_open_repository : sdai_repository;
  ;
END_FUNCTION; -- macro_get_open_repository
(*
```

Argument definitions:

result: (output) An open repository.

6.2.149 macro_get_schema_instance

This macro function shall return a schema instance whose native schema is the ESTS schema. If needed, both a session and the repository the schema instance belongs to shall be opened. Also, if level 3 of transactions is supported, then transaction read-write access shall be started. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
*)
FUNCTION macro_get_schema_instance : schema_instance;
   ;
END_FUNCTION; -- macro_get_schema_instance
(*
```

Argument definitions:

result: (output) A schema instance.

6.2.150 macro_get_sdai_model_unset_mode

This macro function shall return an SDAI-model with unset mode, based on the ESTS schema. If needed, both a session and the repository the SDAI-model belongs to shall be opened. Also, if level 3 of transactions is supported, then transaction read-write access shall be started. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_get_sdai_model_unset_mode : sdai_model;
   ;
END_FUNCTION; -- macro_get_sdai_model_unset_mode
(*
```

Argument definitions:

result: (output) An SDAI-model with unset mode.

6.2.151 macro_get_sdai_model_read_only

This macro function shall return an SDAI-model in read-only mode, based on the ESTS schema. If needed, both a session and the repository the SDAI-model belongs to shall be opened. Also, if level 3 of transactions is supported, then transaction read-write access shall be started. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_get_sdai_model_read_only : sdai_model;
   ;
END_FUNCTION; -- macro_get_sdai_model_read_only
(*
```

Argument definitions:

result: (output) An SDAI-model in read-only mode.

6.2.152 macro_get_sdai_model_read_write

This macro function shall return an SDAI-model in read-write mode, based on the ESTS schema. If needed, both a session and the repository the SDAI-model belongs to shall be opened. Also, if level 3 of transactions is supported, then transaction read-write access shall be started. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
*)
FUNCTION macro_get_sdai_model_read_write : sdai_model;
   ;
END_FUNCTION; -- macro_get_sdai_model_read_write
(*
```

Argument definitions:

result: (output) An SDAI-model in read-write mode.

6.2.153 macro_get_sdai_model_read_write_different

This macro function shall return an SDAI-model in read-write mode, based on the ESTS schema that is different than the model specified as a parameter. If needed, both a session and the repository the SDAI-model belongs to shall be opened. Also, if level 3 of transactions is supported, then transaction read-write access shall be started. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_get_sdai_model_read_write_different
    (modl : sdai_model) : sdai_model;
   ;
END_FUNCTION; -- macro_get_sdai_model_read_write_different
(*
```

Argument definitions:

modl: (input) An SDAI-model based on the ESTS schema.
result: (output) An SDAI-model in read-write mode different from the model specified by the first argument.

6.2.154 macro_get_data_dictionary_model

This macro function shall return a data dictionary model. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_get_data_dictionary_model : sdai_model;
   ;
END_FUNCTION; -- macro_get_data_dictionary_model
(*
```

Argument definitions:

result: (output) An SDAI-model for data dictionary.

6.2.155 macro_get_entity_extent

This macro function shall return entity extent for the entity definition submitted. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
*)
FUNCTION macro_get_entity_extent(def : entity_definition) : entity_extent;
   ;
END_FUNCTION; -- macro_get_entity_extent
(*
```

Argument definitions:

def: (input) An entity definition from the ESTS schema.
result: (output) Entity extent for the instances of the entity specified by the first argument.

6.2.156 macro_check_extent_if_populated

This macro function shall return TRUE if entity extent submitted belongs to the 'populated_folders' set of the contents of the SDAI-model specified by the second argument; otherwise it shall return FALSE. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_check_extent_if_populated(extent : entity_extent;
              modl : sdai_model) : BOOLEAN;
   ;
END_FUNCTION; -- macro_check_extent_if_populated
(*
```

Argument definitions:

extent: (input) An entity extent.

modl: (input) SDAI-model, based on the ESTS schema, containing entity extent specified by the first argument.

result: (output) A BOOLEAN variable. The value is TRUE if the specified entity extent is nonempty, and FALSE otherwise.

6.2.157 macro_check_instance_if_values_unset

This macro function shall return TRUE if all values of the entity instance submitted are unset; otherwise it shall return FALSE. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_check_instance_if_values_unset(inst : application_instance)
      : BOOLEAN;
   ;
END_FUNCTION; -- macro_check_instance_if_values_unset
(*
```

Argument definitions:

inst: (input) An entity instance.

result: (output) A BOOLEAN variable. The value is TRUE if the specified entity instance has all values unset, and FALSE otherwise.

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

6.2.158 macro_compare_aggregates

This macro function shall compare the contents of two aggregates. The value TRUE shall be returned if and only if both aggregates contain exactly the same members with the same repetition. The order in which elements are stored in each of the aggregates is ignored. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_compare_aggregates(aggr1 : aggregate_instance;
             aggr2 : BAG [0:?] OF GENERIC) : BOOLEAN;
   ;
END_FUNCTION; -- macro_compare_aggregates
(*
```

Argument definitions:

aggr1: (input) An aggregate of any Express type of GENERIC.

aggr2: (input) An aggregate of type BAG of GENERIC.

result: (output) A BOOLEAN variable. The value is TRUE if the aggregates specified have the same contents, and FALSE otherwise.

6.2.159 macro_convert_primitive_to_aggregate

This macro function shall converts a primitive value into an aggregate value of this primitive. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION macro_convert_primitive_to_aggregate
   (prim : primitive) : aggregate_primitive;
   ;
END_FUNCTION; -- macro_convert_primitive_to_aggregate
(*
```

Argument definitions:

prim: (input) A primitive representing some aggregate.
result: (output) The aggregate described by the primitive given as the first argument.

6.2.160 macro_clear_aggregate

This macro function shall remove or unset all members from the submitted aggregate. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
PROCEDURE macro_clear_aggregate(aggr : aggregate_primitive);
   ;
END_PROCEDURE; -- macro_clear_aggregate
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

Argument definitions:

aggr: (input) An aggregate that shall be cleared.

6.2.161 asp

This utility function shall convert a *GENERIC* value together with an optional list of defined types into a corresponding assignable_primitive. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
FUNCTION asp
  (the_value : GENERIC; select_specification : LIST[1:?] OF defined_type)
   : assignable_primitive;
  ;
END_FUNCTION;
(*
```

Argument definitions:

result: (output) Value in the form of assignable primitive.

6.2.162 assert

The *assert* procedure is a verdict criterion operation. The verdict criterion is passed if the value of the BOOLEAN parameter is TRUE; otherwise the verdict criteria failed. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
PROCEDURE assert(bool : BOOLEAN);
  ;
END_PROCEDURE; -- assert
(*
```

Argument definitions:

bool: (input) BOOLEAN value submitted for evaluation.

6.2.163 check_instance

The *check_instance* procedure is a verdict criterion operation. The verdict criterion is passed if instance1 and instance2 are value equal as defined in clause 12.2.1.7 of ISO 10303-11. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
PROCEDURE check_instance
  (instance1 : application_instance; instance2 : application_instance);
  ;
END_PROCEDURE;
(*
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

6.2.164 atc

This procedure writes the test case identifier into the conformance log. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
PROCEDURE atc(identifier : STRING);
  ;
END_PROCEDURE;
(*
```

Argument definitions:

identifier: (input) The test case identifier to be written into conformance log.

6.2.165 purpose

This procedure writes the description of a test purpose into the conformance log. The verdict criteria of all subsequent statements after this procedure call till the call to the verdict procedure shall be summarised. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
PROCEDURE purpose(description : STRING);
  ;
END_PROCEDURE;
(*
```

Argument definitions:

description: (input) Test purpose definition to be written into conformance log.

6.2.166 verdict

The *verdict* procedure determines the test verdict of a test purpose by comparing all verdict criteria of the actual test purpose. If all verdict criteria results in TRUE, the test verdict PASS shall be written to the conformance log; otherwise the test verdict FAIL shall be written to the conformance log. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

```
*)
PROCEDURE verdict;
  ;
END_PROCEDURE;
(*
```

6.2.167 print

This auxiliary procedure prints out the string submitted to the conformance log. An executable test suite shall provide an implementation of this function.

EXPRESS specification:

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
*)
PROCEDURE print(str : STRING);
   ;
END_PROCEDURE;

END_SCHEMA;
(*
```

Argument definitions:

str: (input) A string to be printed.

7 SDAI operation schema

This clause specifies the SDAI operation schema that defines the **abstract test operations** as a wrapper around the SDAI operations. The abstract test operations are EXPRESS procedures and functions, intended to invoke the SDAI operations in an encapsulated environment together with the input and output parameter of the SDAI operation and the expected error code and error bases for testing. The abstract test operations are **verdict criterion operation** for positive and negative tests on SDAI operations. In case of a positive test no error is expected during the execution of an SDAI operation (see the NO_ERROR constant in 7.1). In case of a negative test the wrapped SDAI operation is expected to generate the provided error code and error base when executed.

Executable test operations shall be derived from the abstract test operations, by invoking the SDAI operations with the specified parameters and comparing the resulting error code and error base with the expected values afterwards. If both, the resulting and expected error code match and the resulting and expected error base match the value of verdict criterion is TRUE; otherwise it is FALSE.

The following EXPRESS specification begins the **SDAI operation schema** schema.

EXPRESS specification:

```
*)
SCHEMA SDAI_operation_schema;

USE FROM SDAI_dictionary_schema ( -- ISO 10303-22
   attribute,
   explicit_attribute,
   defined_type,
   global_rule,
   named_type,
   uniqueness_rule,
   where_rule);

USE FROM SDAI_session_schema;  -- ISO 10303-22

USE FROM SDAI_parameter_data_schema ( -- ISO 10303-22
   aggregate_instance,
   aggregate_primitive,
   application_instance,
   array_instance,
   assignable_primitive,
   entity_instance,
   iterator,
   list_instance,
   non_persistent_list_instance,
   ordered_collection,
   primitive,
   unordered_collection);
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved

```
(*
```

NOTE 1 The schemas referenced above are specified in the following parts of ISO 10303:

SDAI_dictionary_schema	ISO 10303-22
SDAI_session_schema	ISO 10303-22
SDAI_parameter_data_schema	ISO 10303-22

NOTE 2 Clause 8.3 of ISO 10303-22 provides alternative type definitions for schema_definition and entity_definition for SDAI implementations without a data dictionary. Since the abstract SDAI_operation_schema is defined by using the SDAI_dictionary_schema the alternative definitions shall not be used.

NOTE 3 When transforming the abstract test operations into executable test operations, it may be needed to change the type of parameters from SDAI_dictionary_schema types into string types.

NOTE 4 For testing early binding SDAI implementations the executable test operations may need to select one of several executable SDAI operations, depending of the specified entity type and attribute.

7.1 SDAI operation schema constant definitions

EXPRESS specification:

```
*)
CONSTANT
   NO_ERROR : INTEGER := 0;
   SS_OPN   : INTEGER := 10;
   SS_NAVL  : INTEGER := 20;
   SS_NOPN  : INTEGER := 30;
   RP_NEXS  : INTEGER := 40;
   RP_NAVL  : INTEGER := 50;
   RP_OPN   : INTEGER := 60;
   RP_NOPN  : INTEGER := 70;
   RP_DUP   : INTEGER := 75;
   TR_EAB   : INTEGER := 80;
   TR_EXS   : INTEGER := 90;
   TR_NAVL  : INTEGER := 100;
   TR_RW    : INTEGER := 110;
   TR_NRW   : INTEGER := 120;
   TR_NEXS  : INTEGER := 130;
   MO_NDEQ  : INTEGER := 140;
   MO_NEXS  : INTEGER := 150;
   MO_NVLD  : INTEGER := 160;
   MO_DUP   : INTEGER := 170;
   MX_NRW   : INTEGER := 180;
   MX_NDEF  : INTEGER := 190;
   MX_RW    : INTEGER := 200;
   MX_RO    : INTEGER := 210;
   SD_NDEF  : INTEGER := 220;
   ED_NDEF  : INTEGER := 230;
   ED_NDEQ  : INTEGER := 240;
   ED_NVLD  : INTEGER := 250;
   RU_NDEF  : INTEGER := 260;
   EX_NSUP  : INTEGER := 270;
   AT_NVLD  : INTEGER := 280;
   AT_NDEF  : INTEGER := 290;
   SI_DUP   : INTEGER := 300;
```

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

```
   SI_NEXS   : INTEGER := 310;
   EI_NEXS   : INTEGER := 320;
   EI_NAVL   : INTEGER := 330;
   EI_NVLD   : INTEGER := 340;
   EI_NEXP   : INTEGER := 350;
   SC_NEX    : INTEGER := 360;
   SC_EXS    : INTEGER := 370;
   AI_NEXS   : INTEGER := 380;
   AI_NVLD   : INTEGER := 390;
   AI_NSET   : INTEGER := 400;
   VA_NVLD   : INTEGER := 410;
   VA_NEXS   : INTEGER := 420;
   VA_NSET   : INTEGER := 430;
   VT_NVLD   : INTEGER := 440;
   IR_NEXS   : INTEGER := 450;
   IR_NSET   : INTEGER := 460;
   IX_NVLD   : INTEGER := 470;
   ER_NSET   : INTEGER := 480;
   OP_NVLD   : INTEGER := 490;
   FN_NAVL   : INTEGER := 500;
   SY_ERR    : INTEGER := 1000;
END_CONSTANT;
(*
```

7.2 SDAI operation schema function and procedure definitions

This clause wraps most of the SDAI operations by corresponding EXPRESS functions and procedures. The parameters of the EXPRESS functions and procedures follow the input and output parameters as defined in clause 10 of ISO 10303-22. Two additional parameters for the expected *error_code* and the expected *error_base* are added to the EXPRESS functions and procedures. The provided values for error code and error base are defined in Table 2 of ISO 10303-22.

The following SDAI operations are not mapped to EXPRESS functions or procedures and no abstract test cases are defined for them:

— *record error*, specified in clause 10.4.1 of ISO 10303-22;

— *start event recording*, specified in clause 10.4.2 of ISO 10303-22;

— *stop event recording*, specified in clause 10.4.3 of ISO 10303-22;

— *add to scope*, specified in clause 10.8.1 of ISO 10303-22;

— *is scope owner*, specified in clause 10.8.2 of ISO 10303-22;

— *get scope*, specified in clause 10.8.3 of ISO 10303-22;

— *remove from scope*, specified in clause 10.8.4 of ISO 10303-22;

— *add to export list*, specified in clause 10.8.5 of ISO 10303-22;

— *remove from export list*, specified in clause 10.8.6 of ISO 10303-22;

— *scoped delete*, specified in clause 10.8.7 of ISO 10303-22;

— *scoped copy*, specified in clause 10.8.8 of ISO 10303-22;

— *validate scope reference restrictions*, specified in clause 10.8.9 of ISO 10303-22;

STANDARDSISO.COM : Click to view the full PDF of ISO/TS 10303-35:2003

© ISO 2003 — All rights reserved