Nederlandse

Correctieblad

norm

NEN-ISO/IEC 8652/C1 (en)

Information technology - Programming languages - Ada (ISO/IEC 8652:2012/Cor 1:2016,IDT)

februari 2016
ICS 35.060

Als Nederlands correctieblad is aanvaard:

- ISO/IEC 8652:2012/Cor 1:2016 IDT

Normcommissie 381022 "Programming languages, their environments and system software interfaces"

NEN

©2016 Nederlands Normalisatie-instituut
Postbus 5059, 2600 GB Delft
Telefoon (015) 2 690 390, Fax (015) 2 690 190

INTERNATIONAL STANDARD ISO/IEC 8652:2012
TECHNICAL CORRIGENDUM 1

Published 2016-02-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Programming languages — Ada

TECHNICAL CORRIGENDUM 1

Langages de programmation — Ada

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to ISO/IEC 8652:2012 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

ICS 35.060 **Ref. No. ISO/IEC 8652:2012/Cor.1:2016(E)**

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, SC22, *Programming languages, their environments and system software interfaces*.

Technical Corrigendum 1 cancels and replaces those portions of International Standard ISO/IEC 8652:2012 as specified by the body of this corrigendum. Those portions of the International Standard not modified by this corrigendum remain in force.

Introduction

This corrigendum contains corrections to the Ada 2012 standard [ISO/IEC 8652:2012].

The corrigendum is organized by clauses corresponding to those in the Ada 2012 standard. These clauses include wording changes to the Ada 2012 standard. Subclause headings are given for each subclause that contains a wording change. Other subclauses are omitted. For each change, a reference to the defect report(s) that prompted the wording change is included in the form [8652/0000]. The defect reports have been developed by the ISO/IEC JTC 1/SC 22/WG 9 Ada Rapporteur Group to address specific questions about the Ada standard. Refer to the defect reports for details on the issues.

For each change, an *anchor* paragraph from the original Ada 2012 standard is given. New or revised text and instructions are given with each change. The anchor paragraph can be replaced or deleted, or text can be inserted before or after it. When a heading immediately precedes the anchor paragraph, any text inserted before the paragraph is intended to appear under the heading.

Typographical conventions:

Instructions about the text changes are in this font. The actual text changes are in the same fonts as the Ada 2012 standard - this font for text, this font for syntax, `and this font for Ada source code.`

Introduction

Of International Standard ISO/IEC 8652:2012. Modifications of this section of that standard are found here.

Replace paragraph 57.15: [8652/0117]

- The concept of assertions introduced in the 2005 edition is extended with the ability to specify preconditions and postconditions for subprograms, and invariants for private types. The concept of constraints in defining subtypes is supplemented with subtype predicates that enable subsets to be specified other than as simple ranges. These properties are all indicated using aspect specifications. See subclauses 3.2.4, 6.1.1, and 7.3.2.

by:

- The concept of assertions introduced in the 2005 edition is extended with the ability to specify preconditions and postconditions for subprograms, and invariants for private types and interfaces. The concept of constraints in defining subtypes is supplemented with subtype predicates that enable subsets to be specified other than as simple ranges. These properties are all indicated using aspect specifications. See subclauses 3.2.4, 6.1.1, and 7.3.2.

Replace paragraph 57.16: [8652/0117]

- New forms of expressions are introduced. These are if expressions, case expressions, quantified expressions, and expression functions. As well as being useful for programming in general by avoiding the introduction of unnecessary assignments, they are especially valuable in conditions and invariants since they avoid the need to introduce auxiliary functions. See subclauses 4.5.7, 4.5.8, and 6.8. Membership tests are also made more flexible. See subclauses 4.4 and 4.5.2.

by:

- New forms of expressions are introduced. These are if expressions, case expressions, quantified expressions, expression functions, and raise expressions. As well as being useful for programming in general by avoiding the introduction of unnecessary assignments, they are especially valuable in conditions and invariants since they avoid the need to introduce auxiliary functions. See subclauses 4.5.7, 4.5.8, 6.8, and 11.3. Membership tests are also made more flexible. See subclauses 4.4 and 4.5.2.

ISO/IEC 8652:2012/Cor.1:2016

Section 1: General

1.1 Scope

Replace paragraph 3: [8652/0118]

The language provides rich support for real-time, concurrent programming, and includes facilities for multicore and multiprocessor programming. Errors can be signaled as exceptions and handled explicitly. The language also covers systems programming; this requires precise control over the representation of data and access to system-dependent properties. Finally, a predefined environment of standard packages is provided, including facilities for, among others, input-output, string manipulation, numeric elementary functions, and random number generation, and definition and use of containers.

by:

The language provides rich support for real-time, concurrent programming, and includes facilities for multicore and multiprocessor programming. Errors can be signaled as exceptions and handled explicitly. The language also covers systems programming; this requires precise control over the representation of data and access to system-dependent properties. Finally, a predefined environment of standard packages is provided, including facilities for, among others, input-output, string manipulation, numeric elementary functions, random number generation, and definition and use of containers.

1.1.2 Structure

Replace paragraph 24: [8652/0118]

Each section is divided into subclauses that have a common structure. Each clause and subclause first introduces its subject. After the introductory text, text is labeled with the following headings:

by:

Each clause is divided into subclauses that have a common structure. Each clause and subclause first introduces its subject. After the introductory text, text is labeled with the following headings:

Section 2: Lexical Elements

No changes in this clause.

ISO/IEC 8652:2012/Cor.1:2016

Section 3: Declarations and Types

3.2.4 Subtype Predicates

Replace paragraph 4: [8652/0119; 8652/0120]

- For a (first) subtype defined by a derived type declaration, the predicates of the parent subtype and the progenitor subtypes apply.

by:

- For a (first) subtype defined by a type declaration, any predicates of parent or progenitor subtypes apply.

Delete paragraph 6: [8652/0119]

The *predicate* of a subtype consists of all predicate specifications that apply, and-ed together; if no predicate specifications apply, the predicate is True (in particular, the predicate of a base subtype is True).

Replace paragraph 12: [8652/0120]

- If a subtype is defined by a derived type declaration that does not include a predicate specification, then predicate checks are enabled for the subtype if and only if predicate checks are enabled for at least one of the parent subtype and the progenitor subtypes;

by:

- If a subtype is defined by a type declaration that does not include a predicate specification, then predicate checks are enabled for the subtype if and only if any predicate checks are enabled for parent or progenitor subtypes;

Insert after paragraph 14: [8652/0121]

- Otherwise, predicate checks are disabled for the given subtype.

the new paragraphs:

For a subtype with a directly-specified predicate aspect, the following additional language-defined aspect may be specified with an aspect_specification (see 13.1.1):

Predicate_Failure
: This aspect shall be specified by an expression, which determines the action to be performed when a predicate check fails because a directly-specified predicate aspect of the subtype evaluates to False, as explained below.

Name Resolution Rules

The expected type for the Predicate_Failure expression is String.

Replace paragraph 17: [8652/0122]

- a membership test whose simple_expression is the current instance, and whose membership_choice_list meets the requirements for a static membership test (see 4.9);

by:

- a membership test whose *tested*_simple_expression is the current instance, and whose membership_choice_list meets the requirements for a static membership test (see 4.9);

Replace paragraph 20: [8652/0120]

- a call to a predefined boolean logical operator, where each operand is predicate-static;

by:

- a call to a predefined boolean operator **and**, **or**, **xor**, or **not**, where each operand is predicate-static;

Insert before paragraph 30: [8652/0119]

If predicate checks are enabled for a given subtype, then:

the new paragraphs:

If any of the above Legality Rules is violated in an instance of a generic unit, Program_Error is raised at the point of the violation.

To determine whether a value *satisfies the predicates* of a subtype *S*, the following tests are performed in the following order, until one of the tests fails, in which case the predicates are not satisfied and no further tests are performed, or all of the tests succeed, in which case the predicates are satisfied:

- the value is first tested to determine whether it satisfies any constraints or any null exclusion of *S*;
- then:
  - if *S* is a first subtype, the value is tested to determine whether it satisfies the predicates of the parent and progenitor subtypes (if any) of *S* (in an arbitrary order);
  - if *S* is defined by a subtype_indication, the value is tested to determine whether it satisfies the predicates of the subtype denoted by the subtype_mark of the subtype_indication;
- finally, if *S* is defined by a declaration to which one or more predicate specifications apply, the predicates are evaluated (in an arbitrary order) to test that all of them yield True for the given value.

Replace paragraph 31: [8652/0121; 8652/0119]

On every subtype conversion, the predicate of the target subtype is evaluated, and a check is performed that the predicate is True. This includes all parameter passing, except for certain parameters passed by reference, which are covered by the following rule: After normal completion and leaving of a subprogram, for each **in out** or **out** parameter that is passed by reference, the predicate of the subtype of the actual is evaluated, and a check is performed that the predicate is True. For an object created by an object_declaration with no explicit initialization expression, or by an uninitialized allocator, if any subcomponents have default_expressions, the predicate of the nominal subtype of the created object is evaluated, and a check is performed that the predicate is True. Assertions.Assertion_Error is raised if any of these checks fail.

by:

On every subtype conversion, a check is performed that the operand satisfies the predicates of the target subtype. This includes all parameter passing, except for certain parameters passed by reference, which are covered by the following rule: After normal completion and leaving of a subprogram, for each **in out** or **out** parameter that is passed by reference, a check is performed that the value of the parameter satisfies the predicates of the subtype of the actual. For an object created by an object_declaration with no explicit initialization expression, or by an uninitialized allocator, if any subcomponents have default_expressions, a check is performed that the value of the created object satisfies the predicates of the nominal subtype.

If any of the predicate checks fail, Assertion_Error is raised, unless the subtype whose directly-specified predicate aspect evaluated to False also has a directly-specified Predicate_Failure aspect. In that case, the specified Predicate_Failure expression is evaluated; if the evaluation of the Predicate_Failure expression propagates an exception occurrence, then this occurrence is propagated for the failure of the predicate check; otherwise, Assertion_Error is raised, with an associated message string defined by the value of the Predicate_Failure expression. In the absence of such a Predicate_Failure aspect, an implementation-defined message string is associated with the Assertion_Error exception.

Delete paragraph 32: [8652/0119]

A value *satisfies* a predicate if the predicate is True for that value.

Delete paragraph 33: [8652/0119]

If any of the above Legality Rules is violated in an instance of a generic unit, Program_Error is raised at the point of the violation.

Insert after paragraph 35: [8652/0121; 8652/0119]

6 A Static_Predicate, like a constraint, always remains True for all objects of the subtype, except in the case of uninitialized variables and other invalid values. A Dynamic_Predicate, on the other hand, is checked as specified above, but can become False at other times. For example, the predicate of a record subtype is not checked when a subcomponent is modified.

the new paragraphs:

7 No predicates apply to the base subtype of a scalar type; every value of a scalar type *T* is considered to satisfy the predicates of *T*'Base.

8 Predicate_Failure expressions are never evaluated during the evaluation of a membership test (see 4.5.2) or Valid attribute (see 13.9.2).

9 A Predicate_Failure expression can be a raise_expression (see 11.3).

Examples

```
subtype Basic_Letter is Character -- See A.3.2 for "basic letter".
   with Static_Predicate => Basic_Letter in 'A'..'Z' | 'a'..'z' | 'Æ' | 'æ' | 'Ð' | 'ð'
| 'Þ' | 'þ' | 'ß';

subtype Even_Integer is Integer
   with Dynamic_Predicate => Even_Integer mod 2 = 0,
        Predicate_Failure => "Even_Integer must be a multiple of 2";
```

Text_IO (see A.10.1) could have used predicates to describe some common exceptional conditions as follows:

```
with Ada.IO_Exceptions;
package Ada.Text_IO is

   type File_Type is limited private;

   subtype Open_File_Type is File_Type
      with Dynamic_Predicate => Is_Open (Open_File_Type),
           Predicate_Failure => raise Status_Error with "File not open";
   subtype Input_File_Type is Open_File_Type
      with Dynamic_Predicate => Mode (Input_File_Type) = In_File,
           Predicate_Failure => raise Mode_Error with "Cannot read file: " &
              Name (Input_File_Type);
   subtype Output_File_Type is Open_File_Type
      with Dynamic_Predicate => Mode (Output_File_Type) /= In_File,
           Predicate_Failure => raise Mode_Error with "Cannot write file: " &
              Name (Output_File_Type);

   ...

   function Mode (File : in Open_File_Type) return File_Mode;
   function Name (File : in Open_File_Type) return String;
   function Form (File : in Open_File_Type) return String;

   ...

   procedure Get (File : in Input_File_Type; Item : out Character);

   procedure Put (File : in Output_File_Type; Item : in Character);

   ...

   -- Similarly for all of the other input and output subprograms.
```

3.5 Scalar Types

Insert after paragraph 55: [8652/0123]

For the evaluation of a call on S'Value for an enumeration subtype S, if the sequence of characters of the parameter (ignoring leading and trailing spaces) has the syntax of an enumeration literal and if it corresponds to a literal of the type of S (or corresponds to the result of S'Image for a value of the type), the result is the corresponding enumeration value; otherwise, Constraint_Error is raised. For a numeric subtype S, the evaluation of a call on S'Value with *Arg* of type String is equivalent to a call on S'Wide_Wide_Value for a corresponding *Arg* of type Wide_Wide_String.

the new paragraphs:

For a prefix X that denotes an object of a scalar type (after any implicit dereference), the following attributes are defined:

X'Wide_Wide_Image

X'Wide_Wide_Image denotes the result of calling function S'Wide_Wide_Image with *Arg* being X, where S is the nominal subtype of X.

X'Wide_Image

X'Wide_Image denotes the result of calling function S'Wide_Image with *Arg* being X, where S is the nominal subtype of X.

X'Image

Bestelformulier

NEN

Stuur naar:

NEN Standards Products & Services
t.a.v. afdeling Klantenservice
Antwoordnummer 10214
2600 WB Delft

NEN Standards Products & Services

Postbus 5059
2600 GB Delft

Vlinderweg 6
2623 AX Delft

T (015) 2 690 390
F (015) 2 690 271

www.nen.nl/normshop

Ja, ik bestel

__ ex. NEN-ISO/IEC 8652:2012/C1:2016 en € 0.00

Wilt u deze norm in PDF-formaat? Deze bestelt u eenvoudig via www.nen.nl/normshop

Gratis e-mailnieuwsbrieven

Wilt u op de hoogte blijven van de laatste ontwikkelingen op het gebied van normen, normalisatie en regelgeving? Neem dan een gratis abonnement op een van onze e-mailnieuwsbrieven. www.nen.nl/nieuwsbrieven

Gegevens

Bedrijf / Instelling

T.a.v. O M O V

E-mail

Klantnummer NEN

Uw ordernummer BTW nummer

Postbus / Adres

Postcode Plaats

Telefoon Fax

Factuuradres (indien dit afwijkt van bovenstaand adres)

Postbus / Adres

Postcode Plaats

Datum Handtekening

Retourneren

Fax: (015) 2 690 271
E-mail: klantenservice@nen.nl
Post: NEN Standards Products & Services,
t.a.v. afdeling Klantenservice
Antwoordnummer 10214,
2600 WB Delft
(geen postzegel nodig).

Voorwaarden

- De prijzen zijn geldig tot 31 december 2016, tenzij anders aangegeven.
- Alle prijzen zijn excl. btw, verzend- en handelingskosten en onder voorbehoud bij o.m. ISO- en IEC-normen.
- Bestelt u via de normshop een pdf, dan betaalt u geen handeling en verzendkosten.
- Meer informatie: telefoon (015) 2 690 391, dagelijks van 8.30 tot 17.00 uur.
- Wijzigingen en typefouten in teksten en prijsinformatie voorbehouden.
- U kunt onze algemene voorwaarden terugvinden op: www.nen.nl/leveringsvoorwaarden.

Normalisatie: de wereld op één lijn.