INTERNATIONAL STANDARD

ISO/IEC 8652

Second edition
1995-02-15

AMENDMENT 1
2007-03-15

Information technology — Programming languages — Ada

AMENDMENT 1

Technologies de l'information — Langages de programmation — Ada

AMENDEMENT 1

Reference number
ISO/IEC 8652:1995/Amd.1:2007(E)

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

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

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

Amendment 1 to ISO/IEC 8652:1995 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Amendment 1 cancels and replaces those portions of ISO/IEC 8652:1995 as corrected by Technical Corrigendum 1 ISO/IEC 8652:1995/Cor.1:2001 as specified by the body of this amendment. Those portions of the International Standard as corrected by Technical Corrigendum 1 not modified by this amendment remain in force.

The main emphasis of Amendment 1 is to improve the object-oriented programming and the real-time features of ISO/IEC 8652:1995 while also strengthening reliability.

As in ISO/IEC 8652:1995, Annexes A to J form an integral part of the International Standard as amended. Annexes K to Q are for information only.

Introduction

International Standard ISO/IEC 8652:1995 defines the Ada programming language.

This amendment modifies Ada by making changes and additions that improve:

- The safety of applications written in Ada;
- The portability of applications written in Ada;
- Interoperability with other languages and systems; and
- Accessibility and ease of transition from idioms in other programming and modeling languages.

This amendment incorporates the following major additions to the International Standard:

- Support for the entire ISO/IEC 10646:2003 character repertoire, both in program text and executing programs (see subclauses 2.1, 3.5.2, 3.6.3, A.1, A.3, and A.4);
- Interfaces, to provide a limited form of multiple inheritance of operations (see 3.9.4);
- Improvements for access types, such as null excluding subtypes (see 3.10), additional uses for anonymous access types (see 3.6 and 8.5.1), and anonymous access-to-subprogram subtypes to support “downward closures” (see 3.10 and 3.10.2);
- Additional context clause capabilities: limited views to allow mutually dependent types (see 3.10.1 and 10.1.2) and private with clauses that apply only in the private part of a package (see 10.1.2);
- Aggregates, constructor functions and constants for limited types (see 4.3.1, 6.5, and 7.5);
- Control of overriding to eliminate errors (see 8.3);
- Additional standard packages, including time management (see 9.6), file directory and name management (see A.16), containers (see A.18), execution-time clocks (see D.14), timing events (see D.15), and vector and matrix operations (see G.3);
- A mechanism for writing C unions to make interfaces with C systems easier (see B.3.3);
- New task dispatching policies, including non-preemptive (see D.2.4) and earliest deadline first (see D.2.6); and
- The Ravenscar profile to provide a simplified tasking system for high-integrity systems (see D.13).

This amendment is organized by sections corresponding to those in the International Standard. These sections include wording changes and additions to the International Standard. Clause and subclause headings are given for each clause that contains a wording change. Clauses and subclauses that do not contain any change or addition are omitted.

For each change, an *anchor* paragraph from the International Standard (as corrected by Technical Corrigendum 1) 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 International Standard – this font for text, this font for syntax, `and this font for Ada source code.`

Information technology — Programming languages — Ada

AMENDMENT 1

Introduction

Of International Standard ISO/IEC 8652:1995. Modifications of this section of that International Standard are found here.

Replace paragraph 3:

- Rationale for the Ada Programming Language -- 1995 edition, which gives an introduction to the new features of Ada, and explains the rationale behind them. Programmers should read this first.

by:

- Ada 95 Rationale. This gives an introduction to the new features of Ada incorporated in the 1995 edition of this Standard, and explains the rationale behind them. Programmers unfamiliar with Ada 95 should read this first.
- Ada 2005 Rationale. This gives an introduction to the changes and new features in Ada 2005 (compared with the 1995 edition), and explains the rationale behind them. Programmers should read this rationale before reading this Standard in depth.

Replace paragraph 5:

- The Annotated Ada Reference Manual (AARM). The AARM contains all of the text in the RM95, plus various annotations. It is intended primarily for compiler writers, validation test writers, and others who wish to study the fine details. The annotations include detailed rationale for individual rules and explanations of some of the more arcane interactions among the rules.

by:

- The Annotated Ada Reference Manual (AARM). The AARM contains all of the text in the consolidated Ada Reference Manual, plus various annotations. It is intended primarily for compiler writers, validation test writers, and others who wish to study the fine details. The annotations include detailed rationale for individual rules and explanations of some of the more arcane interactions among the rules.

Replace paragraph 6:

Ada was originally designed with three overriding concerns: program reliability and maintenance, programming as a human activity, and efficiency. This revision to the language was designed to provide greater flexibility and extensibility, additional control over storage management and synchronization, and standardized packages oriented toward supporting important application areas, while at the same time retaining the original emphasis on reliability, maintainability, and efficiency.

by:

Ada was originally designed with three overriding concerns: program reliability and maintenance, programming as a human activity, and efficiency. The 1995 revision to the language was designed to provide greater flexibility and extensibility, additional control over storage management and synchronization, and standardized packages oriented toward supporting important application areas, while at the same time retaining the original emphasis on reliability, maintainability, and efficiency. This amended version provides further flexibility and adds more standardized packages within the framework provided by the 1995 revision.

Replace paragraph 32:

An enumeration type defines an ordered set of distinct enumeration literals, for example a list of states or an alphabet of characters. The enumeration types Boolean, Character, and Wide_Character are predefined.

by:

An enumeration type defines an ordered set of distinct enumeration literals, for example a list of states or an alphabet of characters. The enumeration types Boolean, Character, Wide_Character, and Wide_Wide_Character are predefined.

Replace paragraph 34:

Composite types allow definitions of structured objects with related components. The composite types in the language include arrays and records. An array is an object with indexed components of the same type. A record is an object with named components of possibly different types. Task and protected types are also forms of composite types. The array types String and Wide_String are predefined.

by:

Composite types allow definitions of structured objects with related components. The composite types in the language include arrays and records. An array is an object with indexed components of the same type. A record is an object with named components of possibly different types. Task and protected types are also forms of composite types. The array types String, Wide_String, and Wide_Wide_String are predefined.

Insert after paragraph 38:

From any type a new type may be defined by derivation. A type, together with its derivatives (both direct and indirect) form a derivation class. Class-wide operations may be defined that accept as a parameter an operand of any type in a derivation class. For record and private types, the derivatives may be extensions of the parent type. Types that support these object-oriented capabilities of class-wide operations and type extension must be tagged, so that the specific type of an operand within a derivation class can be identified at run time. When an operation of a tagged type is applied to an operand whose specific type is not known until run time, implicit dispatching is performed based on the tag of the operand.

the new paragraph:

Interface types provide abstract models from which other interfaces and types may be composed and derived. This provides a reliable form of multiple inheritance. Interface types may also be implemented by task types and protected types thereby enabling concurrent programming and inheritance to be merged.

Replace paragraph 41:

Representation clauses can be used to specify the mapping between types and features of an underlying machine. For example, the user can specify that objects of a given type must be represented with a given number of bits, or that the components of a record are to be represented using a given storage layout. Other features allow the controlled use of low level, nonportable, or implementation-dependent aspects, including the direct insertion of machine code.

by:

Aspect clauses can be used to specify the mapping between types and features of an underlying machine. For example, the user can specify that objects of a given type must be represented with a given number of bits, or that the components of a record are to be represented using a given storage layout. Other features allow the controlled use of low level, nonportable, or implementation-dependent aspects, including the direct insertion of machine code.

Replace paragraph 42:

The predefined environment of the language provides for input-output and other capabilities (such as string manipulation and random number generation) by means of standard library packages. Input-output is supported for values of user-defined as well as of predefined types. Standard means of representing values in display form are also provided. Other standard library packages are defined in annexes of the standard to support systems with specialized requirements.

by:

The predefined environment of the language provides for input-output and other capabilities by means of standard library packages. Input-output is supported for values of user-defined as well as of predefined types. Standard means of representing values in display form are also provided.

The predefined standard library packages provide facilities such as string manipulation, containers of various kinds (vectors, lists, maps, etc.), mathematical functions, random number generation, and access to the execution environment.

The specialized annexes define further predefined library packages and facilities with emphasis on areas such as real-time scheduling, interrupt handling, distributed systems, numerical computation, and high-integrity systems.

Replace paragraph 44:

This International Standard replaces the first edition of 1987. In this edition, the following major language changes have been incorporated:

by:

This amended International Standard updates the edition of 1995 which replaced the first edition of 1987. In the 1995 edition, the following major language changes were incorporated.

Replace paragraph 45:

- Support for standard 8-bit and 16-bit character sets. See Section 2, 3.5.2, 3.6.3, A.1, A.3, and A.4.

by:

- Support for standard 8-bit and 16-bit characters was added. See clauses 2.1, 3.5.2, 3.6.3, A.1, A.3, and A.4.

Replace paragraph 46:

- Object-oriented programming with run-time polymorphism. See the discussions of classes, derived types, tagged types, record extensions, and private extensions in clauses 3.4, 3.9, and 7.3. See also the new forms of generic formal parameters that are allowed by 12.5.1, "Formal Private and Derived Types" and 12.7, "Formal Packages".

by:

- The type model was extended to include facilities for object-oriented programming with dynamic polymorphism. See the discussions of classes, derived types, tagged types, record extensions, and private extensions in clauses 3.4, 3.9, and 7.3. Additional forms of generic formal parameters were allowed as described in clauses 12.5.1 and 12.7.

Replace paragraph 47:

- Access types have been extended to allow an access value to designate a subprogram or an object declared by an object declaration (as opposed to just a heap-allocated object). See 3.10.

by:

- Access types were extended to allow an access value to designate a subprogram or an object declared by an object declaration as opposed to just an object allocated on a heap. See clause 3.10.

Replace paragraph 48:

- Efficient data-oriented synchronization is provided via protected types. See Section 9.

by:

- Efficient data-oriented synchronization was provided by the introduction of protected types. See clause 9.4.

Replace paragraph 49:

- The library units of a library may be organized into a hierarchy of parent and child units. See Section 10.

by:

- The library structure was extended to allow library units to be organized into a hierarchy of parent and child units. See clause 10.1.

Replace paragraph 50:

- Additional support has been added for interfacing to other languages. See Annex B.

by:

- Additional support was added for interfacing to other languages. See Annex B.

Replace paragraph 51:

- The Specialized Needs Annexes have been added to provide specific support for certain application areas:

by:

- The Specialized Needs Annexes were added to provide specific support for certain application areas:

Replace paragraph 57:

- Annex H, "Safety and Security"

by:

- Annex H, "High Integrity Systems"

Amendment 1 modifies the 1995 International Standard by making changes and additions that improve the capability of the language and the reliability of programs written in the language. In particular, the changes were designed to improve the portability of programs, interfacing to other languages, and both the object-oriented and real-time capabilities.

The following significant changes with respect to the 1995 edition are incorporated:

- Support for program text is extended to cover the entire ISO/IEC 10646:2003 repertoire. Execution support now includes the 32-bit character set. See clauses 2.1, 3.5.2, 3.6.3, A.1, A.3, and A.4.
- The object-oriented model has been improved by the addition of an interface facility which provides multiple inheritance and additional flexibility for type extensions. See clauses 3.4, 3.9, and 7.3. An alternative notation for calling operations more akin to that used in other languages has also been added. See clause 4.1.3.
- Access types have been further extended to unify properties such as the ability to access constants and to exclude null values. See clause 3.10. Anonymous access types are now permitted more freely and anonymous access-to-subprogram types are introduced. See clauses 3.3, 3.6, 3.10, and 8.5.1.
- The control of structure and visibility has been enhanced to permit mutually dependent references between units and finer control over access from the private part of a package. See clauses 3.10.1 and 10.1.2. In addition, limited types have been made more useful by the provision of aggregates, constants, and constructor functions. See clauses 4.3, 6.5, and 7.5.
- The predefined environment has been extended to include additional time and calendar operations, improved string handling, a comprehensive container library, file and directory management, and access to environment variables. See clauses 9.6.1, A.4, A.16, A.17, and A.18.
- Two of the Specialized Needs Annexes have been considerably enhanced:
  - The Real-Time Systems Annex now includes the Ravenscar profile for high-integrity systems, further dispatching policies such as Round Robin and Earliest Deadline First, support for timing events, and support for control of CPU time utilization. See clauses D.2, D.13, D.14, and D.15.
  - The Numerics Annex now includes support for real and complex vectors and matrices as previously defined in ISO/IEC 13813:1998 plus further basic operations for linear algebra. See clause G.3.
- The overall reliability of the language has been enhanced by a number of improvements. These include new syntax which detects accidental overloading, as well as pragmas for making assertions and giving better control over the suppression of checks. See clauses 6.1, 11.4.2, and 11.5.

Section 1: General

1.1.2 Structure

Replace paragraph 13:

- Annex H, ``Safety and Security"

by:

- Annex H, ``High Integrity Systems"

1.1.4 Method of Description and Syntax Notation

Replace paragraph 9:

return_statement ::= **return** [expression];

return_statement ::= **return**; | **return** expression;

by:

simple_return_statement ::= **return** [expression];

simple_return_statement ::= **return**; | **return** expression;

Insert after paragraph 14:

- If the name of any syntactic category starts with an italicized part, it is equivalent to the category name without the italicized part. The italicized part is intended to convey some semantic information. For example *subtype*_name and *task*_name are both equivalent to name alone.

the new paragraph:

The delimiters, compound delimiters, reserved words, and numeric_literals are exclusively made of the characters whose code position is between 16#20# and 16#7E#, inclusively. The special characters for which names are defined in this International Standard (see 2.1) belong to the same range. For example, the character E in the definition of exponent is the character whose name is "LATIN CAPITAL LETTER E", not "GREEK CAPITAL LETTER EPSILON".

Insert before paragraph 15:

A *syntactic category* is a nonterminal in the grammar defined in BNF under "Syntax." Names of syntactic categories are set in a different font, like_this.

the new paragraph:

When this International Standard mentions the conversion of some character or sequence of characters to upper case, it means the character or sequence of characters obtained by using locale-independent full case folding, as defined by documents referenced in the note in section 1 of ISO/IEC 10646:2003.

1.2 Normative References

Replace paragraph 3:

ISO/IEC 1539:1991, *Information technology — Programming languages — FORTRAN*.

by:

ISO/IEC 1539-1:2004, *Information technology — Programming languages — Fortran — Part 1: Base language*.

Replace paragraph 4:

ISO 1989:1985, *Programming languages — COBOL*.

by:

ISO/IEC 1989:2002, *Information technology — Programming languages — COBOL*.

Insert after paragraph 5:

ISO/IEC 6429:1992, *Information technology — Control functions for coded graphic character sets.*

the new paragraph:

ISO 8601:2004, *Data elements and interchange formats — Information interchange — Representation of dates and times*.

Replace paragraph 7:

ISO/IEC 9899:1990, *Programming languages — C*.

by:

ISO/IEC 9899:1999, *Programming languages — C*, supplemented by Technical Corrigendum 1:2001 and Technical Corrigendum 2:2004.

Replace paragraph 8:

ISO/IEC 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*, supplemented by Technical Corrigendum 1:1996.

by:

ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*.

ISO/IEC 14882:2003, *Programming languages — C++*.

ISO/IEC TR 19769:2004, *Information technology — Programming languages, their environments and system software interfaces — Extensions for the programming language C to support new character data types*.

1.3 Definitions

Replace paragraph 1:

Terms are defined throughout this International Standard, indicated by *italic* type. Terms explicitly defined in this International Standard are not to be presumed to refer implicitly to similar terms defined elsewhere. Terms not defined in this International Standard are to be interpreted according to the *Webster's Third New International Dictionary of the English Language*. Informal descriptions of some terms are also given in Annex N, "Glossary".

by:

Terms are defined throughout this International Standard, indicated by *italic* type. Terms explicitly defined in this International Standard are not to be presumed to refer implicitly to similar terms defined elsewhere. Mathematical terms not defined in this International Standard are to be interpreted according to the *CRC Concise Encyclopedia of Mathematics, Second Edition*. Other terms not defined in this International Standard are to be interpreted according to the *Webster's Third New International Dictionary of the English Language*. Informal descriptions of some terms are also given in Annex N, "Glossary".

Section 2: Lexical Elements

2.1 Character Set

Replace paragraph 1:

The only characters allowed outside of comments are the graphic_characters and format_effectors.

by:

The character repertoire for the text of an Ada program consists of the entire coding space described by the ISO/IEC 10646:2003 Universal Multiple-Octet Coded Character Set. This coding space is organized in *planes*, each plane comprising 65536 characters.

Delete paragraph 2:

character ::= graphic_character | format_effector | other_control_function

Replace paragraph 3:

graphic_character ::= identifier_letter | digit | space_character | special_character

by:

A character is defined by this International Standard for each cell in the coding space described by ISO/IEC 10646:2003, regardless of whether or not ISO/IEC 10646:2003 allocates a character to that cell.

Replace paragraph 4:

The character repertoire for the text of an Ada program consists of the collection of characters called the Basic Multilingual Plane (BMP) of the ISO 10646 Universal Multiple-Octet Coded Character Set, plus a set of format_effectors and, in comments only, a set of other_control_functions; the coded representation for these characters is implementation defined (it need not be a representation defined within ISO-10646-1).

by:

The coded representation for characters is implementation defined (it need not be a representation defined within ISO/IEC 10646:2003). A character whose relative code position in its plane is 16#FFFE# or 16#FFFF# is not allowed anywhere in the text of a program.

The semantics of an Ada program whose text is not in Normalization Form KC (as defined by section 24 of ISO/IEC 10646:2003) is implementation defined.

Replace paragraph 5:

The description of the language definition in this International Standard uses the graphic symbols defined for Row 00: Basic Latin and Row 00: Latin-1 Supplement of the ISO 10646 BMP; these correspond to the graphic symbols of ISO 8859-1 (Latin-1); no graphic symbols are used in this International Standard for characters outside of Row 00 of the BMP. The actual set of graphic symbols used by an implementation for the visual representation of the text of an Ada program is not specified.

by:

The description of the language definition in this International Standard uses the character properties General Category, Simple Uppercase Mapping, Uppercase Mapping, and Special Case Condition of the documents referenced by the note in section 1 of ISO/IEC 10646:2003. The actual set of graphic symbols used by an implementation for the visual representation of the text of an Ada program is not specified.

Replace paragraph 6:

The categories of characters are defined as follows:

by:

Characters are categorized as follows:

Delete paragraph 7:

identifier_letter
upper_case_identifier_letter | lower_case_identifier_letter

Replace paragraph 8:

upper_case_identifier_letter
Any character of Row 00 of ISO 10646 BMP whose name begins "Latin Capital Letter".

by:

letter_uppercase
Any character whose General Category is defined to be "Letter, Uppercase".

Replace paragraph 9:

lower_case_identifier_letter
Any character of Row 00 of ISO 10646 BMP whose name begins "Latin Small Letter".

by:

letter_lowercase
Any character whose General Category is defined to be "Letter, Lowercase".

letter_titlecase
Any character whose General Category is defined to be "Letter, Titlecase".

letter_modifier
Any character whose General Category is defined to be "Letter, Modifier".

letter_other
Any character whose General Category is defined to be "Letter, Other".

mark_non_spacing
Any character whose General Category is defined to be "Mark, Non-Spacing".

mark_spacing_combining
Any character whose General Category is defined to be "Mark, Spacing Combining".

Replace paragraph 10:

digit
One of the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9.

by:

number_decimal
Any character whose General Category is defined to be "Number, Decimal".

number_letter
Any character whose General Category is defined to be "Number, Letter".

punctuation_connector
Any character whose General Category is defined to be "Punctuation, Connector".

other_format
Any character whose General Category is defined to be "Other, Format".

Replace paragraph 11:

space_character
The character of ISO 10646 BMP named "Space".

by:

separator_space
Any character whose General Category is defined to be "Separator, Space".

Replace paragraph 12:

special_character
Any character of the ISO 10646 BMP that is not reserved for a control function, and is not the space_character, an identifier_letter, or a digit.

by:

separator_line
: Any character whose General Category is defined to be "Separator, Line".

separator_paragraph
: Any character whose General Category is defined to be "Separator, Paragraph".

Replace paragraph 13:

format_effector
: The control functions of ISO 6429 called character tabulation (HT), line tabulation (VT), carriage return (CR), line feed (LF), and form feed (FF).

by:

format_effector
: The characters whose code positions are 16#09# (CHARACTER TABULATION), 16#0A# (LINE FEED), 16#0B# (LINE TABULATION), 16#0C# (FORM FEED), 16#0D# (CARRIAGE RETURN), 16#85# (NEXT LINE), and the characters in categories separator_line and separator_paragraph.

other_control
: Any character whose General Category is defined to be "Other, Control", and which is not defined to be a format_effector.

other_private_use
: Any character whose General Category is defined to be "Other, Private Use".

other_surrogate
: Any character whose General Category is defined to be "Other, Surrogate".

Replace paragraph 14:

other_control_function
: Any control function, other than a format_effector, that is allowed in a comment; the set of other_control functions allowed in comments is implementation defined.

by:

graphic_character
: Any character that is not in the categories other_control, other_private_use, other_surrogate, format_effector, and whose relative code position in its plane is neither 16#FFFE# nor 16#FFFF#.

Replace paragraph 15:

The following names are used when referring to certain special_characters:

by:

The following names are used when referring to certain characters (the first name is that given in ISO/IEC 10646:2003):

Delete paragraph 16:

In a nonstandard mode, the implementation may support a different character repertoire; in particular, the set of characters that are considered identifier_letters can be extended or changed to conform to local conventions.

Replace paragraph 17:

1 Every code position of ISO 10646 BMP that is not reserved for a control function is defined to be a graphic_character by this International Standard. This includes all code positions other than 0000 - 001F, 007F - 009F, and FFFE - FFFF.

by:

1 The characters in categories other_control, other_private_use, and other_surrogate are only allowed in comments.

2.2 Lexical Elements, Separators, and Delimiters

Replace paragraph 2:

The text of a compilation is divided into *lines*. In general, the representation for an end of line is implementation defined. However, a sequence of one or more format_effectors other than character tabulation (HT) signifies at least one end of line.

by:

The text of a compilation is divided into *lines*. In general, the representation for an end of line is implementation defined. However, a sequence of one or more format_effectors other than the character whose code position is 16#09# (CHARACTER TABULATION) signifies at least one end of line.

Replace paragraph 3:

In some cases an explicit *separator* is required to separate adjacent lexical elements. A separator is any of a space character, a format effector, or the end of a line, as follows:

by:

In some cases an explicit *separator* is required to separate adjacent lexical elements. A separator is any of a separator_space, a format_effector, or the end of a line, as follows:

Replace paragraph 4:

- A space character is a separator except within a comment, a string_literal, or a character_literal.

by:

- A separator_space is a separator except within a comment, a string_literal, or a character_literal.

Replace paragraph 5:

- Character tabulation (HT) is a separator except within a comment.

by:

- The character whose code position is 16#09# (CHARACTER TABULATION) is a separator except within a comment.

Replace paragraph 8:

A *delimiter* is either one of the following special characters

by:

A *delimiter* is either one of the following characters:

2.3 Identifiers

Replace paragraph 2:

```
identifier ::=
  identifier_letter {[underline] letter_or_digit}
```

by:

```
identifier ::=
  identifier_start {identifier_start | identifier_extend}
```

Replace paragraph 3:

```
letter_or_digit ::= identifier_letter | digit
```

by:

```
identifier_start ::=
    letter_uppercase
  | letter_lowercase
  | letter_titlecase
  | letter_modifier
  | letter_other
  | number_letter

identifier_extend ::=
    mark_non_spacing
  | mark_spacing_combining
  | number_decimal
  | punctuation_connector
  | other_format
```

Replace paragraph 4:

An identifier shall not be a reserved word.

by:

After eliminating the characters in category other_format, an identifier shall not contain two consecutive characters in category punctuation_connector, or end with a character in that category.

Replace paragraph 5:

All characters of an identifier are significant, including any underline character. Identifiers differing only in the use of corresponding upper and lower case letters are considered the same.

by:

Two identifiers are considered the same if they consist of the same sequence of characters after applying the following transformations (in this order):

- The characters in category other_format are eliminated.
- The remaining sequence of characters is converted to upper case.

Replace paragraph 6:

In a nonstandard mode, an implementation may support other upper/lower case equivalence rules for identifiers, to accommodate local conventions.

by:

After applying these transformations, an identifier shall not be identical to a reserved word (in upper case).

In a nonstandard mode, an implementation may support other upper/lower case equivalence rules for identifiers, to accommodate local conventions.

NOTES

3 Identifiers differing only in the use of corresponding upper and lower case letters are considered the same.

Replace paragraph 8:

```
Count       X     Get_Symbol    Ethelyn    Marion

Snobol_4    X1    Page_Count    Store_Next_Item
```

by:

```
Count       X     Get_Symbol    Ethelyn    Marion
Snobol_4    X1    Page_Count    Store_Next_Item
Πλάτων            -- Plato
Чайковский        -- Tchaikovsky
θ   φ             -- Angles
```