TECHNICAL REPORT

ISO/IEC TR 24717

First edition
2009-08-01

Information technology — Programming languages, their environments and system software interfaces — Collection classes for programming language COBOL

Technologies de l'information — Langages de programmation, leurs environnements et interfaces du logiciel système — Classes de collection pour le langage de programmation COBOL

Reference number
ISO/IEC TR 24717:2009(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

Contents

Page

Tables

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.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

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.

ISO/IEC TR 24717, which is a Technical Report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*, in collaboration with INCITS Technical Committee J4, Programming language COBOL.

Introduction

This Technical Report specifies an object-oriented class library for managing collections of object references — a collection class library.

This Technical Report extends the COBOL specification defined in ISO/IEC 1989:2002, *Information technology — Programming languages — COBOL* by providing classes to manage collections.

Annex A forms a normative part of this Technical Report. Annex B, Annex C, Annex D, and the Bibliography are for information only.

Information technology — Programming languages, their environments and system software interfaces — Collection classes for programming language COBOL

1 Scope

This Technical Report specifies the interfaces and behavior of a common class library for managing sets of object references in COBOL. The purpose of this Technical Report is to promote a high degree of portability in implementations of the class library, even though some elements are subject to trial before completion of a final design suitable for standardization.

This specification builds on the syntax and semantics defined in ISO/IEC 1989:2002.

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

3 Conformance

This Technical Report is based on ISO/IEC 1989:2002. Conformance to this Technical Report does not require a full implementation of ISO/IEC 1989:2002. The interaction of the features of this Technical Report with features that are not provided by an implementation of ISO/IEC 1989:2002 is processor dependent.

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

4.1
collection
set of object references managed by an instance of a collection class

4.2
iterator
object that allows sequencing through all of the object references managed by an instance of a collection class

5 Description techniques

Description techniques and language fundamentals are the same as those described in ISO/IEC 1989:2002. Additionally the class diagrams in Appendix D are presented using Unified Modeling Language (UML).

6 Changes to ISO/IEC 1989:2002

These changes refer to clause and rule numbers in ISO/IEC 1989:2002.

1. 16.1, Base class, insert into BaseFactoryInterface interface definition after 'Procedure Division.'

"

```
    Method-id. ClassName.
    Data division.
    Linkage section.
    01  outName pic n(31).
    Procedure division returning outName.
    End method ClassName.
*>
    Method-id. ExternalClassName.
    Data division.
    Linkage section.
    01  outName.
        02  outNameCategory pic x.
            88  ExternalClassNameDisplay value "X".
            88  ExternalClassNameNational value "N".
        02  outNameNational pic n(1024).
        02  outNameDisplay redefines outNameNational
            pic x(1024).
    Procedure division returning outName.
    End method ExternalClassName.
*>
```

"

2. Add new sections prior to 16.1.1, renumbering 16.1.1 to 16.1.2, etc.

"

16.1.1 ClassName

The ClassName method is a factory method that returns the internal name of the class associated with the factory for which it is invoked.

16.1.1.1 General Rules

1) The ClassName method returns the internal class name of the factory in outName.

NOTE 1 The class name is returned in national characters and might not be suitable for use where an exact case-sensitive class name is required.

NOTE 2 The following code can be used to determine the name of the class of the instance object anObject:

```
Invoke anObject "FactoryObject" returning aFactoryObject
Invoke aFactoryObject "ClassName" returning aClassName
```

16.1.2 ExternalClassName

The ExternalClassName method is a factory method that returns the external name of the class associated with the factory for which it is invoked.

16.1.2.1 General Rules

1) If the external class name specified in the AS clause of the CLASS-ID paragraph is a national literal, the ExternalClassName method sets ExternalClassNameNational to true, and returns the external class name of the factory in outNameNational. Otherwise the external class name of the factory is returned in outNameDisplay and ExternalClassNameDisplay is set to true.

"

3. Add new section.

"**16.2 ExceptionInformation class**

The ExceptionInformation class provides information on exceptions that may occur in any method that inherits from the BASE class.

```
Interface-id. ExceptionInterface
  Inherits BaseInterface.
Environment division.
Configuration section.
Repository.
    Interface BaseInterface.
*>
Procedure division.
    Method-id. Get Property ExceptionClassName.
    Data division.
    Linkage section.
    01  outName pic n any length.
    Procedure division returning outName.
    End method ExceptionClassName.
*>
    Method-id. Set Property ExceptionClassName.
    Data division.
    Linkage section.
    01  outName pic n any length.
    Procedure division using outName.
    End method ExceptionClassName.
*>
    Method-id. Get Property ExceptionCode.
    Data division.
    Linkage section.
    01  outCode pic n(31).
    Procedure division returning outCode.
    End method ExceptionCode.
*>
    Method-id. Set Property ExceptionCode.
    Data division.
    Linkage section.
    01  outCode pic n(31).
    Procedure division using outCode.
    End method ExceptionCode.
*>
    Method-id. Get Property ExceptionMethodName.
    Data division.
    Linkage section.
    01  outName pic n any length.
    Procedure division returning outName.
    End method ExceptionMethodName.
*>
    Method-id. Set Property ExceptionMethodName.
    Data division.
    Linkage section.
    01  outName pic any length.
    Procedure division using outName.
    End method ExceptionMethodName.
*>
    Method-id. Get Property ExceptionMessage.
    Data division.
    Linkage section.
    01  outMessage pic n any length.
    Procedure division returning outMessage.
    End method ExceptionMessage.
*>
```

```
    Method-id. Set Property ExceptionMessage.
    Data division.
    Linkage section.
    01  outMessage pic n any length.
    Procedure division using outMessage.
    End method ExceptionMessage.
*>
    Method-id. Get Property ExceptionSourceObject.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject.
    End method ExceptionSourceObject.
*>
    Method-id. Set Property ExceptionSourceObject.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division using outObject.
    End method ExceptionSourceObject.
*>
End Interface ExceptionInterface.
```

Properties of the ExceptionInformation class may be used to set or get:

- the method where the exception object was raised,
- the class that contained that method,
- the object on which that method was invoked,
- an exception code that is indicative of the exception that occurred,
- a description of the exception that occurred.

16.2.1 ExceptionClassName property

The ExceptionClassName property is used to set or get the name of the class in which the exception was raised. The maximum size of the method name returned is 1024.

16.2.2 ExceptionCode property

The ExceptionCode property is used to set or get a national character string that indicates the type of exception raised. All exception codes defined by this standard begin with the characters “EO-“.

NOTE This method is overridden by exception classes that are specific to and associated with the classes that raise the exceptions. Valid exception codes are defined in each class to correspond to those exceptions raised by that class. Exception codes defined for a subclass are in addition to those defined by the class from which the subclass inherits.

16.2.3 ExceptionMethodName property

The ExceptionMethodName property is used to set or get the name of the method in which the exception was raised.

16.2.4 ExceptionMessage property

The ExceptionMessage property is used to set or get a message describing why the exception was raised. The contents of the ExceptionMessage are implementor-defined.

16.2.5 ExceptionSourceObject property

The ExceptionSourceObject property sets or gets an instance object reference or a factory object reference. The object reference returned references the object upon which the method that raised the exception object was invoked.

"

7 COBOL Collection Classes

Programmers working with objects quickly find it necessary to manage references to multiple instances of related objects. When managed, these references make up a collection. Collections in COBOL are managed by a collection class as specified in this Technical Report.

Object references are inserted into an instance of the collection class according to the attributes specified when the instance of the collection is created. If a collection is not ordered and not keyed, the object references are inserted sequentially within the collection in the order that they are added. If a collection is ordered, the object references are added using a method of the ordered collection class to position that object reference within the collection relative to the other object references of the collection. If a collection is keyed, each object reference in the collection is associated with a key item when it is added to the collection. Keys are shortcuts to allow retrieval of object references from a collection using a convenient name rather than an object reference. Keys associated with object references do not specify the order of those object references within a collection. If a collection is sorted, a method within each member of the collection is invoked as the object reference is added to the collection. This method returns a national data item that is used to control the sequence in which object references are returned from the collection.

Object references may be retrieved from collections: sequentially; directly using their ordinal positions or, for keyed collections, using their keys; or indirectly using an instance of the Iterator class. The Iterator class provides for retrieval of object references from a collection either in the order they exist within a collection, or in a sequence defined at the creation of the iterator. The iterator may also be used to delete an object from the underlying collection. Collections that contain no object references are empty collections. Attempting to retrieve an object reference from an empty collection will raise an exception object.

Object references that are inserted into or retrieved from instances of the collection class may be restricted to object references that conform to a particular class by using the parameterized forms of the collection class interfaces. Other than these restrictions on the class of the object reference, the methods of these parameterized interfaces function equivalently to the methods of non-parameterized interfaces. These parameterized interfaces, however, do not inherit from the common collection-class interface, but redefine the methods of the collection class within each subclass. The functionalities of these methods are equivalent to the functionalities of the corresponding methods defined in the collection class interface.

When exceptions occur in the invocation of the methods of the collection class, an exception object is created and raised. The exception object contains information that describes the exception and the method in which the exception occurred.

7.1 Collection class

The Collection class is the base class for all collections and includes all the methods necessary to manage the membership of a collection.

The instance methods of the Collection class perform the following functions:

- add object references to a collection
- delete object references from a collection
- compare the membership of two collection instances
- copy an instance of a collection
- count object references in a collection
- return object references from a collection
- create an iterator for a collection

The following is the specification of the formal interfaces supported by the Collection class.

7.1.1 Collection instance interface

The methods associated with the instance objects of the Collection class provide the facilities necessary to establish and maintain a collection.

An instance of the Collection class maintains the ordinal position for each object reference that is added to that collection class. The ordinal position of the first object reference is 1. Object references may be retrieved in sequence or directly by their ordinal position. The collection maintains the ordinal position of the last object reference retrieved from or added to the collection. This object reference is the current reference for this collection.

Collection instance interface

```
Interface-id. CollectionInterface
  Inherits BaseInterface.
Environment division.
Configuration section.
Repository.
    Interface BaseInterface
    Interface CollectionExInterface
    Interface IteratorInterface.
*>
Procedure division.
    Method-id. AddObject.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    Procedure division using by value inObject
      raising CollectionExInterface.
    End method AddObject.
*>
    Method-id. CompareCollection.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    01  outBoolean usage bit pic 1.
    Procedure division using by value inObject returning outBoolean.
    End method CompareCollection.
*>
    Method-id. CopyCollection.
    Data division.
    Linkage section.
    01  outObject usage object reference active-class.
    Procedure division returning outObject.
    End method CopyCollection.
*>
    Method-id. CountObjects.
    Data division.
    Linkage section.
    01  outBinary usage binary-long.
    Procedure division returning outBinary.
    End method CountObjects.
*>
    Method-id. CreateIterator.
    Data division.
    Linkage section.
    01  inSequence pic N any length.
    01  outIterator usage object reference IteratorInterface.
    Procedure division using optional inSequence returning outIterator
      raising CollectionExInterface.
    End method CreateIterator.
*>
    Method-id. DeleteAll.
    Procedure division.
    End method DeleteAll.
*>
```

```
    Method-id. DeleteCurrent.
    Procedure division
      raising CollectionExInterface.
    End method DeleteCurrent.
*>
    Method-id. DeleteObject.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    Procedure division using by value inObject
      raising CollectionExInterface.
    End method DeleteObject.
*>
    Method-id. Exists.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    01  outBoolean usage bit pic 1.
    Procedure division using by value inObject returning outBoolean.
    End method Exists.
*>
    Method-id. Ordinal.
    Data division.
    Linkage section.
    01  outOrdinal usage binary-long.
    Procedure division returning outOrdinal.
    End method Ordinal.
*>
    Method-id. ReturnCurrent.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExInterface.
    End method ReturnCurrent.
*>
    Method-id. ReturnFirst.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExInterface.
    End method ReturnFirst.
*>
    Method-id. ReturnLast.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExInterface.
    End method ReturnLast.
*>
    Method-id. ReturnNext.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExInterface.
    End method ReturnNext.
*>
    Method-id. ReturnObject.
    Data division.
    Linkage section.
    01  inOrdinal usage binary-long.
    01  outObject usage object reference.
    Procedure division using by value inOrdinal returning outObject
      raising CollectionExInterface.
    End method ReturnObject.
```

```
*>
    Method-id. ReturnPrevious.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExInterface.
    End method ReturnPrevious.
*>
End Interface CollectionInterface.
```

Parameterized collection instance interface

```
Interface-id. ParamCollectionInterface
  Inherits BaseInterface
  Using ParamClass.
Environment division.
Configuration section.
Repository.
    Interface BaseInterface
    Interface CollectionExInterface
    Interface IteratorInterface
    Class     ParamClass.
*>
Procedure division.
    Method-id. AddObject.
    Data division.
    Linkage section.
    01  inObject usage object reference ParamClass.
    Procedure division using by value inObject
      raising CollectionExInterface.
    End method AddObject.
*>
    Method-id. CompareCollection.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    01  outBoolean usage bit pic 1.
    Procedure division using by value inObject returning outBoolean.
    End method CompareCollection.
*>
    Method-id. CopyCollection.
    Data division.
    Linkage section.
    01  outObject usage object reference active-class.
    Procedure division returning outObject.
    End method CopyCollection.
*>
    Method-id. CountObjects.
    Data division.
    Linkage section.
    01  outBinary usage binary-long.
    Procedure division returning outBinary.
    End method CountObjects.
*>
    Method-id. CreateIterator.
    Data division.
    Linkage section.
    01  inSequence pic N any length.
    01  outIterator usage object reference IteratorInterface.
    Procedure division using optional inSequence returning outIterator
      raising CollectionExInterface.
    End method CreateIterator.
*>
    Method-id. DeleteAll.
    Procedure division.
    End method DeleteAll.
*>
```

```
    Method-id. DeleteCurrent.
    Procedure division
      raising CollectionExInterface.
    End method DeleteCurrent.
*>
    Method-id. DeleteObject.
    Data division.
    Linkage section.
    01  inObject usage object reference paramClass.
    Procedure division using by value inObject
      raising CollectionExInterface.
    End method DeleteObject.
*>
    Method-id. Exists.
    Data division.
    Linkage section.
    01  inObject usage object reference paramClass.
    01  outBoolean usage bit pic 1.
    Procedure division using by value inObject returning outBoolean.
    End method Exists.
*>
    Method-id. Ordinal.
    Data division.
    Linkage section.
    01  outOrdinal usage binary-long.
    Procedure division returning outOrdinal.
    End method Ordinal.
*>
    Method-id. ReturnCurrent.
    Data division.
    Linkage section.
    01  outObject usage object reference paramClass.
    Procedure division returning outObject
      raising CollectionExInterface.
    End method ReturnCurrent.
*>
    Method-id. ReturnFirst.
    Data division.
    Linkage section.
    01  outObject usage object reference paramClass.
    Procedure division returning outObject
      raising CollectionExInterface.
    End method ReturnFirst.
*>
    Method-id. ReturnLast.
    Data division.
    Linkage section.
    01  outObject usage object reference paramClass.
    Procedure division returning outObject
      raising CollectionExInterface.
    End method ReturnLast.
*>
    Method-id. ReturnNext.
    Data division.
    Linkage section.
    01  outObject usage object reference paramClass.
    Procedure division returning outObject
      raising CollectionExInterface.
    End method ReturnNext.
*>
    Method-id. ReturnObject.
    Data division.
    Linkage section.
    01  inOrdinal usage binary-long.
    01  outObject usage object reference paramClass.
    Procedure division using by value inOrdinal returning outObject
      raising CollectionExInterface.
    End method ReturnObject.
```