ISO/WDTR 24717

Date: 2005-02-11

Reference number of document: **WDTR 24717**

J4/05-0042

WG4n0231

Version 1.0

Committee identification: ISO/IEC JTC 1/SC 22 /WG 4

Secretariat: ANSI

Previous Version: J4/05-0007

Information Technology — Programming languages, their environments and system software interfaces — COBOL Collection Classes

Warning

This document is an ISO/IEC proposed draft Technical Report. It is not an ISO/IEC International Technical Report. It is distributed for review and comment. It is subject to change without notice and shall not be referred to as an International Technical Report or International Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: Technical report
Document subtype: 2
Document stage: (2) WD Under Consideration
Document language: E

© ISO/IEC 2004 – All rights reserved

Copyright notice

This ISO/IEC document is a working draft and is copyright-protected by ISO/IEC.

Requests for permission to reproduce this document for the purpose of selling it should be addressed as shown below or to ISO's member body in the country of the requester:

Copyright manager
ISO Central Secretariat
1 rue de Varembé
1211 Geneva 20 Switzerland
tel: +41 22 749 0111
fax: +41 22 734 0179
email: central@iso.ch

Reproduction for sales purposes may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

Contents

© ISO/IEC 2004 – All rights reserved

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.

Technical Reports are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft Technical Reports adopted by the joint technical committee are circulated to national bodies for voting. Publication as an Technical Report requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this proposed draft Technical Report may be the subject of patent rights. Neither ISO nor IEC shall be held responsible for identifying any or all such patent rights.

Proposed draft Technical Report ISO/IEC 24717 was prepared by Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*. INCITS Technical Committee J4, Programming language COBOL, contributed to the development.

Proposed draft Technical Report ISO/IEC 24717 extends the COBOL specification defined in ISO/IEC 1989:2002, Information technology — Programming languages, their environments and system software interfaces — Programming language COBOL by providing classes to manage collections.

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

© ISO/IEC 2004 – All rights reserved

Introduction

This proposed draft Technical Report specifies an object-oriented class library managing collections of object references – A Collection Class Library.

In order to provide as much stability as possible to implementors and users, ISO/IEC JTC 1 Subcommittee 22 intends that the syntax and semantics be changed for purposes of standardization only as necessary to address issues arising in implementation or use of the featured class library.

The purpose of the collection class library is to give programmers the ability to easily manage sets of related objects during program execution.

Information Technology —

Programming languages, their environments and system software interfaces —

COBOL Collection Classes

1 Scope

This proposed draft Technical Report specifies the interface and behavior of a common class library for managing sets of object references in COBOL. The purpose of this proposed draft 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 normative documents contain provisions which, through reference in this text, constitute provisions of this proposed draft Technical Report ISO/IEC 24717. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this proposed draft Technical Report ISO/IEC 24717 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

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

3 Conformance to this proposed draft Technical Report

Conformance to this proposed draft Technical Report requires conformance to ISO/IEC 1989:2002 as specified in clause 3, Conformance to this International Standard, and the normative specifications of this proposed draft Technical Report.

4 Terms and definitions

For the purposes of this proposed draft Technical Report, the following definitions apply.

© ISO/IEC 2004 – All rights reserved

4.1 collection: A set of 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.

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 any length.
       Procedure division returning outName.
       End method ClassName.
   *>
   ```
   "
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 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 class name of the factory in outName.

   2) The maximum size of outName is 1024 national characters.

   NOTE 1 The maximum size of outName is designed to be compatible with non-COBOL external object names.

   NOTE 2 The code below 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.
   ```
   "

3. Add new section.

   "16.2 Exception Class

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

© ISO/IEC 2004 – All rights reserved

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

Methods of the Exception class may be invoked to determine:

- 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 occured
- and a description of the exception that occurred.

16.2.1 ExceptionClassName method

The ExceptionClassName method returns 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 method

The ExceptionCode method returns 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 method

The ExceptionMethodName method returns the name of the method in which the exception was raised.

16.2.4 ExceptionMessage method

The ExceptionMessage method returns a message describing why the exception was raised. The contents of the ExceptionMessage are implementor-defined.

16.2.5 ExceptionSourceObject method

The ExceptionSourceObject method returns an instance or factory object reference. The object reference returned is the object associated with the method that raised the exception object.
"

7 Additions to ISO/IEC 1989:2002

The addition below is intended to be a new normative Annex to ISO/IEC 1989:2002.

© ISO/IEC 2004 – All rights reserved

COBOL Collection Classes

1 Introduction

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 preliminary draft Technical Report.

The COBOL collection class library has the following features:

- Manages object references
- Provides for access to these object references
- Allows copying of collections
- Allows navigation, including in a sorted sequence, of the references managed by these collections via an Iterator class

All object references in a COBOL collection are universal object references. An object view may be used to reference these objects as if they were described as a particular class.

Object references are inserted into instances 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 may be 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 via 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 used to control the sequence in which object references are returned from the collection.

Object references may be retrieved from collections: sequentially; directly via their ordinal positions or, for keyed collections, via 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.

© ISO/IEC 2004 – All rights reserved

2 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.

2.1 Collection Instance Interface

```
Interface-id. CollectionInterface
  Inherits BaseInterface.
Environment division.
Configuration section.
Repository.
    Interface BaseInterface
    Interface CollectionExceptionInterface
    Interface IteratorInterface.
*>
Procedure division.
    Method-id. AddObject.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    Procedure division using by value inObject
      raising CollectionExceptionInterface.
    End method AddObject.
*>
    Method-id. CompareCollection.
    Data division.
    Linkage section.
    01  inObject usage object reference CollectionInterface.
    01  outBoolean usage bit pic 1.
    Procedure division using by value inObject returning outBoolean
      raising CollectionExceptionInterface.
    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.
```

© ISO/IEC 2004 – All rights reserved

```
    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 CollectionExceptionInterface.
    End method CreateIterator.
*>
    Method-id. DeleteAll.
    Procedure division.
    End method DeleteAll.
*>
    Method-id. DeleteCurrent.
    Procedure division
      raising CollectionExceptionInterface.
    End method DeleteCurrent.
*>
    Method-id. DeleteObject.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    Procedure division using by value inObject
      raising CollectionExceptionInterface.
    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  outBinary usage binary-long.
    Procedure division returning outBinary.
    End method Ordinal.
*>
    Method-id. ReturnCurrent.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExceptionInterface.
    End method ReturnCurrent.
*>
    Method-id. ReturnFirst.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExceptionInterface.
    End method ReturnFirst.
*>
    Method-id. ReturnLast.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExceptionInterface.
    End method ReturnLast.
*>
    Method-id. ReturnNext.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExceptionInterface.
    End method ReturnNext.
*>
    Method-id. ReturnObject.
    Data division.
    Linkage section.
    01  inBinary usage binary-long.
    01  outObject usage object reference.
    Procedure division using by value inBinary returning outObject
      raising CollectionExceptionInterface.
    End method ReturnObject.
*>
    Method-id. ReturnPrevious.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising CollectionExceptionInterface.
    End method ReturnPrevious.
*>
End Interface CollectionInterface.
```

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

2.1.1 AddObject method

The AddObject method adds the specified object reference to the collection and assigns the object reference an ordinal position equal to 1 greater than that of the highest ordinal position associated with the collection. If the object reference is NULL, the CollectionException object is raised.

© ISO/IEC 2004 – All rights reserved

NOTE Adding an object reference invalidates any iterators associated with this collection.

2.1.2 CompareCollection method

The CompareCollection method compares the membership of the collection to the membership of another collection and returns either 1 (true) or 0 (false) indicating whether those collections reference the same objects . Key values need not match. Collections that contain duplicate references to an object are equal only if the same number of duplicate references to that object exists in both collections

NOTE The order in which object references were added to the two collections does not matter.

2.1.3 CopyCollection method

The CopyCollection method creates a new collection of the same class with identical content and returns an object reference to the new collection. Related iterators are not copied. The current object of the new collection is the first object of that collection.

NOTE The sequence of the new collection is the same as that of the old collection. If this is a sorted collection, the sequencing method of the new collection is the same as that of the old.

2.1.4 CountObjects method

The CountObject method returns the number of object references managed by this instance of the Collection class. Duplicate references to an object are counted as distinct object references.

2.1.5 CreateIterator method

The CreateIterator method creates and returns an iterator instance for this collection as specified in 6, Iterator class. An optional sequencing method parameter may be specified. If this parameter does not refer to a valid method as defined in 2.2, Sequencing method, the CollectionException object is raised and CreateIterator returns a NULL object reference. If the sequencing method parameter is omitted, object references are returned by the iterator in the sequence that they were added to the collection.

NOTE 1 There may be more than one iterator associated with a collection. These iterators may differ in sequence.

NOTE 2 If the collection is an ordered collection and the sequencing method is omitted, the sequence in which object references are returned by the iterator can differ from the sequence in which object references are returned by the collection.

2.1.6 DeleteAll method

The DeleteAll method removes all object references from the collection.

NOTE Deleting all object references invalidates all iterators associated with this collection.

2.1.7 DeleteCurrent method

The DeleteObject method removes the current reference from the collection. If there is no current reference, a CollectionException object is raised.

The ordinal positions of all object references with ordinal positions greater than the current reference are decremented by 1. The object reference with the ordinal position that becomes equal to the ordinal position of the object reference that has been deleted becomes the current reference. If the ordinal position of the object reference deleted is the highest ordinal position for this collection, there is no current reference.

NOTE Deleting an object reference invalidates all iterators associated with this collection.

2.1.8 DeleteObject method

The DeleteObject method removes the specified object reference from the collection. If the input object reference is not part of this collection, a CollectionException object is raised.

If the specified object reference is a member of the collection more than once, all object references are removed from the collection. For each object reference deleted, the ordinal positions of all object references with ordinal positions greater than that object reference are decremented by 1. If the current reference is deleted, the object reference that occurs next in sequence becomes the current reference.

NOTE Deleting an object reference invalidates all iterators associated with this collection.

2.1.9 Exists method

The Exists method tests to see if the specified object reference is a member of the set of object references managed by the collection class instance. If the object reference is a member of the collection, the returning item is set to 1 (true) otherwise the returning item is set to 0 (false).

2.1.10 Ordinal method

The Ordinal method returns the ordinal position of the current reference. If there is no current reference, this method returns 0.

2.1.11 ReturnCurrent method

The ReturnCurrent method returns the current reference. If there is no current reference, a CollectionException object is raised and the return object reference is set to NULL.

2.1.12 ReturnFirst method

The ReturnFirst returns the object reference with an ordinal position of 1 from the collection. If there are no object references in this collection, a CollectionException object is raised and the return object reference is set to NULL.

2.1.13 ReturnLast method

The ReturnLast returns the object reference with the highest ordinal position from the collection. If there are no object references in this collection, a CollectionException object is raised and the return object reference is set to NULL.

© ISO/IEC 2004 – All rights reserved

2.1.14 ReturnNext method

The ReturnNext returns the object reference with an ordinal position 1 greater than the ordinal position of the current reference. If the current reference is the last object reference in the collection, the return object reference is set to NULL and a CollectionException object is raised. If there is no current reference, ReturnNext returns the first object reference in the collection.

2.1.15 ReturnObject method

The ReturnObject method returns the object reference whose ordinal position within the collection is equal to inBinary. If inBinary is less than one or greater than the number of object references in the collection, the return object is set to NULL and a CollectionException object is raised.

2.1.16 ReturnPrevious method

The ReturnPrevious returns the object reference with an ordinal position 1 less than the ordinal position of the current reference. If the current reference is the first object in the collection, the return object is set to NULL and a CollectionException object is raised. If there is no current reference, ReturnPrevious returns the last object reference in the collection.

2.2 Sequencing method

Sequencing methods are used by the CreateIterator method of the Collection class and by the NewSortedCollection method of the SortedCollection class to determine the sequence in which the object references are returned by the resultant iterator or sorted collection. If the sequencing methods for two object references return data items that compare equal, those object references are returned in the sequence that they were added to the collection. Rules for comparison are specified in 8.8.4.1.1, Relation conditions, in ISO/IEC 1989-2002 Programming Language COBOL.

```
Method-id. method-name.
Data division.
Linkage section.
01  outString pic N any length.
Procedure division returning outString.
End method method-name.
```

Method-name shall be the name that is passed as a parameter to the CreateIterator method or to the NewSortedCollection method of the SortedCollection. Each object in the collection shall define a method with the name method-name. That method shall conform to the above specification.

The maximum length for OutString is 1024 national characters.

3 OrderedCollection Class

The OrderedCollection class inherits from the Collection Class. The OrderedCollection class contains additional methods for inserting objects relative to other objects within the collection. Unlike the base Collection class, adding object references to an OrderedCollection using the methods of the OrderedCollection class may change the ordinal position of other object references within the collection.

NOTE Adding an object reference invalidates any iterators associated with this collection.

Object references may be added to an OrderedCollection using the AddObject method of the base Collection class. This would be equivalent to using the AddLast method of the OrderedCollection class.

3.1 OrderedCollection Instance Interface

```
Interface-id. OrderedCollectionInterface
  Inherits CollectionInterface.
Environment division.
Configuration section.
Repository.
    Interface CollectionInterface
    Interface OrderedCollectionExceptionInterface.
*>
Procedure division.
    Method-id. AddAfter.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    Procedure division using by value inObject
      raising OrderedCollectionExceptionInterface.
    End method AddAfter.
*>
    Method-id. AddBefore.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    Procedure division using by value inObject
      raising OrderedCollectionExceptionInterface.
    End method AddBefore.
*>
    Method-id. AddFirst.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    Procedure division using by value inObject
      raising OrderedCollectionExceptionInterface.
    End method AddFirst.
*>
    Method-id. AddLast.
    Data division.
    Linkage section.
    01  inObject usage object reference.
```

© ISO/IEC 2004 – All rights reserved

```
    Procedure division using by value inObject
      raising OrderedCollectionExceptionInterface.
    End method AddLast.
*>
End Interface OrderedCollectionInterface.
```

3.1.1 AddAfter method

The AddAfter method adds the object reference to the collection and assigns the object reference an ordinal position equal to 1 greater than the ordinal position of the current reference. The ordinal position of any object reference with an ordinal position greater than the ordinal position of the current reference is incremented by 1. If there is no current reference, a CollectionException object is raised and the object reference is not added to the collection. If the object reference is NULL, a CollectionException object is raised.

3.1.2 AddBefore method

The AddBefore method adds the object reference to the collection and assigns the object reference an ordinal position equal to the ordinal position of the current reference. The ordinal position of the current reference and of any object reference with an ordinal position greater than the ordinal position of the current reference is incremented by 1. If there is no current reference, a CollectionException object is raised and the object reference is not added to the collection. If the object reference is NULL, a CollectionException object is raised.

3.1.3 AddFirst method

The AddFirst method adds the object reference to the collection and assigns the object reference an ordinal position equal to 1. The ordinal position of all other object references within the collection is incremented by 1. If there are no object references in the collection, the object reference is added to the collection. If the object reference is NULL, a CollectionException object is raised.

3.1.4 AddLast method

The AddLast method adds the object reference to the collection and assigns the object reference an ordinal position equal to 1 greater than the ordinal position of the highest ordinal position associated with collection. If there are no object references in the collection, the ordinal position assigned to the object reference added is 1. If the object reference is NULL, a CollectionException object is raised.

4 KeyedCollection Class

The KeyedCollection class inherits from the Collection class.

The KeyedCollection class associates a national data item with each object reference that is a member of the collection. An object reference may be returned from the collection by invoking the ReturnKeyedObject method passing the object reference's associated national data item as an argument.

Object references may not be added to a KeyedCollection using the AddObject method of the base Collection class.

4.1 KeyedCollection Instance Interface

```
Interface-id. KeyedCollectionInterface
  Inherits CollectionInterface.
Environment division.
Configuration section.
Repository.
    Interface CollectionInterface
    Interface KeyedCollectionExceptionInterface.
*>
Procedure division.
    Method-id. AddKeyed.
    Data division.
    Linkage section.
    01  inObject usage object reference.
    01  inKey pic N any length.
    Procedure division using by value inObject
      by reference inKey
      raising KeyedCollectionExceptionInterface.
    End method AddKeyed.
*>
    Method-id. ReturnKeyFromCurrent.
    Data division.
    Linkage section.
    01  outKey pic N any length.
    Procedure division returning outKey
      raising KeyedCollectionExceptionInterface.
    End method ReturnKeyFromCurrent.
*>
    Method-id. ReturnKeyFromOrdinal.
    Data division.
    Linkage section.
    01  inBinary usage binary-long.
    01  outKey pic N any length.
    Procedure division using by value inBinary returning outKey
      raising KeyedCollectionExceptionInterface.
    End method ReturnKeyFromOrdinal.
*>

    Method-id. ReturnKeyedObject.
    Data division.
    Linkage section.
```

© ISO/IEC 2004 – All rights reserved

```
    01  inKey pic N any length.
    01  outObject usage object reference.
    Procedure division using inKey returning outObject
      raising KeyedCollectionExceptionInterface.
    End method ReturnKeyedObject.
*>
End Interface KeyedCollectionInterface.
```

4.1.1 AddKeyed method

The AddKeyed method is equivalent to the AddObject method of the Collection class, however, the AddKeyed method of the KeyedCollection class requires an additional parameter. This parameter is a national data item that may be used to retrieve the object reference from the collection. Attempting to add an object reference to the collection with a key that duplicates a key that is already part of the collection raises a CollectionException object and the method terminates without altering the collection. All other rules for the AddKeyed method are as specified in 2.2.1, AddObject method.

NOTE The addition of an object reference invalidates all iterators associated with this collection.

4.1.2 ReturnKeyFromCurrent method

The ReturnKeyFromCurrent method returns the specified national data item that was associated with the current reference when that object reference was added to the KeyedCollection. If there is no current reference, a CollectionException object is raised and the returned data item is a zero-length national data item.

4.1.3 ReturnKeyFromOrdinal method

The ReturnKeyFromOrdinal method returns the specified national data item that was associated with the object reference at the ordinal position specified in inBinary. If there is no object reference at the specified ordinal position, a CollectionException object is raised and the returned data item is a zero-length national data item.

4.1.4 ReturnKeyedObject method

The ReturnKeyedObject method returns the object reference that was associated with the specified national data item when that object reference was added to the KeyedCollection. If there is no equivalent national data item associated with a member of the collection, a CollectionException object is raised and the ReturnKeyedObject returned object reference is set to null.

4.2 Overridden methods

These methods are defined in the Collection instance interface and are overridden in a class that implements KeyedCollectionInterface.

4.2.1 AddObject method

The AddObject method of the KeyedCollection class always raises a CollectionException object and the method terminates without altering the collection.

NOTE The override of the AddObject method prevents the addition of object references to a keyed collection without a key. All object references in a keyed collection are added using the AddKeyed method

© ISO/IEC 2004 – All rights reserved

5 SortedCollection Class

The SortedCollection class inherits from the Collection class.

The SortedCollection class sequences object references in a collection by a national character string returned from a sequencing method associated with each object reference in that collection.

The addition of object references to a Sorted Collection can change the ordinal position of other object references within the collection.

5.1 SortedCollection Factory Interface

```
Interface-id. SortedCollectionFactoryInterface
  Inherits CollectionInterface.
Environment division.
Configuration section.
Repository.
    Interface CollectionInterface
    Interface SortedCollectionExceptionInterface.
*>
Procedure division.
    Method-id. NewSortedCollection.
    Data division.
    Linkage section.
    01  inSequence pic N any length.
    01  outCollection usage object reference SortedCollectionInterface.
    Procedure division using inSequence returning outCollection
      raising SortedCollectionExceptionInterface.
    End method NewSortedCollection.
*>
End Enterface SortedCollectionFactoryInterface
```

5.1.1 NewSortedCollection method

The NewSortedCollection method creates and returns a SortedCollection. The invocation of NewSortedCollection shall specify a sequencing method argument. This sequencing method is described in 2.2, Sequencing Method. The name of the sequencing method is passed as an argument to the NewSortedCollection. Each time an object reference is added to a sorted collection, the sequencing method is invoked to obtain the sequencing string. Once a sequencing string has been associated with an object reference in a collection, that sequencing string remains constant as long as that object reference is a member of that collection.

5.2 Overridden Factory methods

The method below is defined in BaseFactoryInterface and is overridden in a class which implements SortedCollectionFactoryInterface.

5.2.1 New Method

The New method of the SortedCollection class returns a NULL object.

5.3 SortedCollection Instance Interface

```
Interface-id. SortedCollectionInterface
  Inherits CollectionInterface.
Environment division.
Configuration section.
Repository.
    Interface CollectionInterface
    Interface SortedCollectionExceptionInterface.
*>
End Interface SortedCollectionInterface.
```

5.4 Overridden instance methods

The method below is defined in CollectionInterface and is overridden in a class that implements the SortedCollectionInterface.

5.4.1 AddObject method

The AddObject method of the SortedCollection class adds the object reference to the collection and assigns the object reference an ordinal position that places the object reference in sequence by the national character string returned by the sequencing method of the object whose reference is being added. The ordinal position of any object reference with an ordinal position greater than the ordinal position of the current reference is incremented by 1. If there is no sequencing method in the object being added, or the method does not conform to the specification in 2.2, Sequencing method, a CollectionException object is raised and the object reference is not added to the collection. If the object reference is NULL, a CollectionException object is raised.

© ISO/IEC 2004 – All rights reserved

6 Iterator Class

An iterator is an object used to access the object references that are part of a collection instance. Iterator instances are created using the Create-Iterator method of a collection. Any change in the membership of the underlying collection, except those changes made by the DeleteCurrent method of the iterator, invalidates that iterator, and any reference to an invalidated raises a CollectionException object.

```
Interface-id. IteratorInterface
  Inherits BaseInterface.
Environment division.
Configuration section.
Repository.
    Interface BaseInterface
    Interface IteratorExceptionInterface.
*>
Procedure division.
    Method-id. DeleteCurrent.
    Procedure division
      raising IteratorExceptionInterface.
    End method DeleteCurrent.
*>
    Method-id. ReturnCurrent.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising IteratorExceptionInterface.
    End method ReturnCurrent.
*>
    Method-id. ReturnFirst.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising IteratorExceptionInterface.
    End method ReturnFirst.
*>
    Method-id. ReturnLast.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising IteratorExceptionInterface.
    End method ReturnLast.
*>
    Method-id. ReturnNext.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising IteratorExceptionInterface.
    End method ReturnNext.
*>
    Method-id. ReturnOrdinal.
    Data division.
    Linkage section.
    01  outBinary usage binary-long.
    Procedure division returning outBinary
      raising IteratorExceptionInterface.
    End method ReturnPrevious.
*>
    Method-id. ReturnPrevious.
    Data division.
    Linkage section.
    01  outObject usage object reference.
    Procedure division returning outObject
      raising IteratorExceptionInterface.
    End method ReturnPrevious.
*>
End Interface IteratorInterface.
```

The iterator accesses the collected object references sequentially as specified in 2.1.5 CreateIterator method. The iterator instance methods that return object references from an iterator return those object references in the sequence specified at the creation of the iterator.

NOTE If there is no sequencing method defined at the creation of the iterator, object references are returned in the order that they were added to the associated collection. If the associated collection is an ordered collection, this sequence may differ from the sequence that would be returned using the methods of the associated collection.

The current reference for an iterator is the last object reference returned by any of those methods. If there are multiple iterators associated with the associated collection, each iterator has its own current reference.

NOTE The current reference for an iterator is not related to the current reference in the associated collection.

If the iterator references a collection that has no associated object references, either because the that collection has no object references or all of the object references have been removed, all invocations of that iterator's instance methods raises a CollectionException object, and any method that returns an object reference returns null

6.1 DeleteCurrent method

The DeleteCurrent method removes the current reference from the associated collection. The next sequential object reference referenced by the iterator becomes the current reference, unless the object reference removed was the last object in sequence referenced by the iterator, in which case there is no current reference. If there is no current reference when the DeleteCurrent method is invoked, a CollectionException object is raised. If the object reference deleted is the current reference for the associated collection, the current reference is set as described in 2.1.7, DeleteCurrent.

NOTE Deleting an object reference invalidates any other iterators associated with this collection.

6.2 ReturnCurrent method

The ReturnCurrent method returns the object reference to the current reference. If there is no current reference when the ReturnCurrent method is invoked, a CollectionException object is raised and the method returns the null object reference value.
.

© ISO/IEC 2004 – All rights reserved

6.3 ReturnFirst method

The ReturnFirst method returns the first object reference in sequence from the iterator. If there are no object references within the collection associated with the iterator, a CollectionException object is raised and the returned object reference is set to null.

6.4 ReturnLast method

The ReturnLast method returns the last object reference in sequence from the iterator. If there are no object references within the collection associated with the iterator, a CollectionException object is raised and the returned object reference is set to null.

6.5 ReturnNext method

The ReturnNext method returns the next reference object in sequence from the iterator. If the current reference is the last sequential object reference referenced by the iterator, a CollectionException object is raised and the returned object reference is set to null.

6.5 ReturnOrdinal method

The ReturnOrdinal method returns the ordinal position in the associated collection of the current reference. If there is no current reference, a CollectionException object is raised and the returned object reference is set to null.

6.6 ReturnPrevious method

The ReturnPrevious method returns the previous object reference in sequence from the iterator. If the current reference is the first sequential object reference referenced by the iterator, a CollectionException object is raised and the returned object reference is set to null.

© ISO/IEC 2004 – All rights reserved

7 Collection Exception Classes

The Collection exception class inherits from the Exception class and provides specific information on exceptions that may occur in the Collection Class methods. Each subclass of the collection class implements a subclass of the Collection Exception class to provide specific returned values for the ExceptionCode method.

7.1 Collection Exception Interfaces

```
Interface-id. CollectionExceptionInterface
  Inherits ExceptionInterface.
Environment division.
Configuration section.
Repository.
    Interface ExceptionInterface.
End CollectionExceptionInterface.

Interface-id. KeyedCollectionExceptionInterface
  Inherits CollectionExceptionInterface.
Environment division.
Configuration section.
Repository.
    Interface CollectionExceptionInterface.
End KeyedCollectionExceptionInterface.

Interface-id. OrderedCollectionExceptionInterface
  Inherits CollectionExceptionInterface.
Environment division.
Configuration section.
Repository.
    Interface CollectionExceptionInterface.
End OrderedCollectionExceptionInterface.

Interface-id. SortedCollectionExceptionInterface
  Inherits CollectionExceptionInterface.
Environment division.
Configuration section.
Repository.
    Interface CollectionExceptionInterface.
End SortedCollectionExceptionInterface.

Interface-id. IteratorExceptionInterface
  Inherits CollectionExceptionInterface.
Environment division.
Configuration section.
Repository.
    Interface CollectionExceptionInterface.
End IteratorExceptionInterface.
```

7.2 ExceptionCode Method Returned Values

The ExceptionCode method returns specific code values for each type of exception that can occur.

© ISO/IEC 2004 – All rights reserved

7.2.1 Collection class ExceptionCode method Returned Values

The Collection class can return the following values:

ExceptionCode Value	Exception
EO-NULL	Object reference is null
EO-BEGINNING-OF-COLLECTION	A ReturnPrevious method was invoked, but the CurrentObject was the first object in the collection or a ReturnObject method was invoked with an ordinal position less than 1
EO-END-OF-COLLECTION	A ReturnNext method was invoked, but the CurrentObject was the last object in the collection or a ReturnObject method was invoked with an ordinal position greater than the ordinal position of the last object reference in the collection
EO-INVALID-SEQUENCING-METHOD	Sequencing method is not a method of one or more object references
EO-NO-CURRENT-OBJECT-REFERENCE	CurrentObject does not reference an object
EO-NOT-IN-COLLECTION	The object reference is not a member of the collection
EO-EMPTY	There are no object references in the collection

7.2.2 OrderedCollection class ExceptionCode method Returned Values

The OrderedCollection class has no additional returned values.

7.2.3 KeyedCollection class ExceptionCode method Returned Values

The KeyedCollection class can return the following additional values:

ExceptionCode Value	Exception
EO-DUPLICATE-KEY	The AddKeyed method was invoked with a key that duplicates a key contained in this collection.
EO-INVALID-KEY	The ReturnKeyedObject method was invoked with a key that this collection does not contain
EO-NO-KEY	The AddObject method was invoked

7.2.4 SortedCollection class ExceptionCode method Returned Values

The SortedCollection class can return the following values:

ExceptionCode Value	Exception
EO-NEW	The New factory method was invoked

7.2.5 Iterator class ExceptionCode method Returned Values

The Iterator class can return the following values:

ExceptionCode Value	**Exception**
EO-INVALIDATED-ITERATOR	The membership of the collection associated with the iterator has changed externally to the iterator

© ISO/IEC 2004 – All rights reserved

Annex A
(normative)

Language element lists

A.1 Implementor-defined element list

The following is a list of the language elements within this proposed draft Technical Report that depend on implementor definition to complete the specification of the elements. Each element is defined as required, optional, or conditionally required. Furthermore, each element is defined as requiring (or not requiring) user documentation. These terms have the following meaning:

Required: The element shall be provided by the implementor. When the element is part of a feature that is optional or processor-dependent, the item is not required if the optional or processor-dependent feature is not implemented.

Optional: The element may be provided at the implementor's option.

Conditionally required: If the associated feature or language element is implemented then this element is also required.

Documentation required: If the element is provided by the implementor, the implementor's user documentation shall document the element or shall reference other documentation that fulfills this requirement.

[1] ExceptionMessage (content when a CollectionException object is raised). This item is required. This item shall be documented in the implementor's user documentation. (6.3 ExceptionMethod method)

A.2 Undefined language element list

The following are language elements within this proposed draft Technical Report that are explicitly undefined.

Annex B
(informative)

Unresolved technical issues

B.1 General

Some technical issues have proven difficult to resolve and have not been addressed in the current design. Those issues are presented here in order to obtain feedback from reviewers and early implementors.

1) The original specification for the COBOL collection classes was for typed collections, that is, collections of objects that were constrained to collect objects that comply to a particular interface. The optimum way of accomplishing this would be to define the Collection class as a parameterized class. However, this conflicted with another design specification, which was to design our collection classes to be able to collect any type of object, including non-Cobol objects and Universal objects. This could not be done with parameterized classes, since we have no class definition that corresponds to the Universal object. The author of this current TR believed that defining the mechanisms to appropriately extend COBOL in this manner would unduly delay this TR.

   The author believes that the following changes to the standard are necessary to define these mechanisms.

   a) Reintroduction of the optional keyword UNIVERSAL for the OBJECT REFERENCE phrase of the USAGE clause.

   b) Creation of syntax to allow the parameters in a parameterized class to be optional. When a parameter is optional, the syntax must provide a default class for the compiler to expand. This class can be UNIVERSAL.

   c) Enhance the syntax of the EXPANDS clause of the repository entry to allow the clause to be omitted if all parameters are optional. Also, enhance the syntax of the EXPANDS clause to handled omitted parameters.

   d) Modifications to the rules of class expansion to properly handle parameter substitution when the parameter is UNIVERSAL. (There are places that substituting UNIVERSAL instead of a class name would produce erroneous code.)

   The author notes that, with the introduction of the ClassName method of the BASE class, a programmer can interrogate any object reference that inherits BASE, and filter the objects added to a collection. The author also notes that the addition of Method Overloading (see #3 below) could allow a certain amount of run time conformance checking and may be an acceptable substitute for typed classes for some applications.

© ISO/IEC 2004 – All rights reserved

2) Currently the keyed collection class does not allow object references to be added to the class without keys. Some other collection class libraries allow both keyed and non-keyed object references within a keyed collection class. Should we allow this?

3) COBOL does not support method overloading. There is a proposal to add this feature to the next standard. If method overloading is added to the next standard, the following methods should be renamed.

Class	Method	Renamed to
KeyedCollectionClass	AddKeyed	AddObject
SortedCollectionClass	NewSortedCollection	New

The old methods should then be either declared archaic or obsolete.

4) Any-length Elementary Items are in the draft for the next standard. When this TR is added to the next standard, or the draft standard becomes the current standard, the sequencing method should be enhanced to use this feature.

Annex C
(informative)

Concepts

C.1 COBOL Collection Classes

Cobol collection classes facilitate the management of sets of related object references. These object references may be of the same class or of different classes. Object references may be added or deleted from a collection instance. Collection instances may be copied and emptied. Object references in collections may be retrieved directly, using an iterator or, in the case of Keyed collections, by key.

C.1.1 Collections

The base class for the collection classes is the Collection. This class provides all of the methods necessary to create and accesses a simple collection.

Example:

Suppose we have a class:

```
*> ACCOUNT CLASS
CLASS-ID. Account INHERITS Base.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS Base.
*> Factory of the Account Class.
*>      This factory supplies a method for creating new accounts
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 number-of-accounts PIC 9(5) VALUE ZERO.
PROCEDURE DIVISION.
METHOD-ID. newAccount.
DATA DIVISION.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.
01 an-object USAGE IS OBJECT REFERENCE ACTIVE-CLASS.
PROCEDURE DIVISION RETURNING an-object.
begin-here.
    INVOKE SELF "new" RETURNING an-object.
    ADD 1 TO number-of-accounts.
    INVOKE an-object "initializeAccount"
        USING BY CONTENT number-of-accounts.
EXIT METHOD.
END METHOD newAccount.
END FACTORY.
*>      Instance definition for the Account Class
```

© ISO/IEC 2004 – All rights reserved

```
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 account-name PIC X(20).
01 account-balance PIC S9(9)V99.
01 account-number PIC X(9).
PROCEDURE DIVISION.
*>     Method to return the account number
METHOD-ID. AccountNumber.
DATA DIVISION.
LINKAGE-STORAGE SECTION.
01 display-number PIC N(9).
PROCEDURE DIVISION RETURNING display-number.
disp-balance.
    MOVE account-number to display-number
EXIT METHOD.
END METHOD AccountNumber.
*>     Method to return the account holders name
METHOD-ID. GetName.
DATA DIVISION.
LINKAGE-STORAGE SECTION.
01 display-name PIC N(20).
PROCEDURE DIVISION RETURNING display-name.
get-name.
    MOVE account-name to display-name
EXIT METHOD.
END METHOD GetName.
*>     Method to set the account holders name
METHOD-ID. SetName.
DATA DIVISION.
LINKAGE-STORAGE SECTION.
01 display-name PIC N(20).
PROCEDURE DIVISION USING display-name.
set-name.
    MOVE display-name to account-name
EXIT METHOD.
END METHOD SetName.
*>     Method called by NewAccount to initialize
*>     the fields of the new account
METHOD-ID. initializeAccount.
DATA DIVISION.
LINKAGE SECTION.
01 new-account-number PIC 9(5).
PROCEDURE DIVISION USING new-account-number.
Begin-initialization.
    MOVE ZERO TO account-balance
    MOVE new-account-number TO account-number
EXIT METHOD.
END METHOD initializeAccount.
END OBJECT
END CLASS Account.
```

To create two accounts and add them to the collection we would need the following:

```
...
WORKING-STORAGE SECTION.
01  an-account usage object reference Account.
01  a-collection usage object reference CollectionInterface.
```

```
...
Procedure Division.
...
*>     Create the Collection
INVOKE Collection "New" RETURNING a-collection.
*>     Create an Account
INVOKE Account "NewAccount" RETURNING an-account.
*>     Add the new account to the collection
INVOKE a-collection "AddObject" USING an-account.
*>     Create another Account
INVOKE Account "NewAccount" RETURNING an-account.
*>     Add the new account to the collection
INVOKE a-collection "AddObject" USING an-account.
```

Object references in collections are kept in sequence as the object references are added. Object references may then be retrieved either sequentially, or by their position in the sequence, with the first object reference being position 1.

Example:

To retrieve the two accounts added above in the sequence we added them, we could do the following:

```
INVOKE a-collection "ReturnFirst" RETURNING an-account.
INVOKE a-collection "ReturnNext" RETURNING an-account.
```

Or, to retrieve the second account directly we could do the following:

```
INVOKE a-collection "ReturnObject" USING 2 RETURNING an-account.
```

Either of the above examples would set the current object reference to account number 2, so:

```
INVOKE a-collection "ReturnCurrent" RETURNING an-account.
```

would retrieve an object reference to account number 2.

C.1.2 Ordered Collections

Ordered collections allow object references to be added at places other than at the end of the sequence. Object references added to an ordered collection may be added to the beginning of the sequence, the end of the sequence, or before or after a specific object in the sequence.

Example:

To create two accounts and add them to the ordered collection in descending sequence we could use the following:

```
...
WORKING-STORAGE SECTION.
01  an-account usage object reference Account.
01  an-ordered-collection usage object reference OrderedCollection.
...
Procedure Division.
...
*>     Create the Collection
INVOKE OrderedCollection "New" RETURNING an-ordered-collection.
```

© ISO/IEC 2004 – All rights reserved

```
*>      Create an Account
INVOKE Account "NewAccount" RETURNING an-account.
*>      Add the new account to the collection
INVOKE an-ordered-collection "AddLast" USING an-account.
*>      Create another Account
INVOKE Account "NewAccount" RETURNING an-account.
*>      Add the new account to the collection prior to
*>      the first account
INVOKE an-ordered-collection "AddFirst" USING an-account.
```

When we retrieve the object references in sequence, account number 2 is retrieved, then account number 1:

```
INVOKE an-ordered-collection "ReturnFirst" RETURNING an-account. *> this
                                                       *> will return
                                                       *> account number 2
INVOKE an-ordered-collection "ReturnNext" RETURNING an-account. *> this
                                                       *> will return
                                                       *> account number 1
```

If we now add an account as follows:

```
INVOKE an-ordered-collection "AddBefore" RETURNING an-account.
```

the sequence of the accounts within the collection is account number 2, then number 3 then number 1.

C.1.3 Keyed collection

Keyed collections associate a key with each object reference added to the collection, allowing an object reference to be returned by specifying the key associated with that object reference. These keys are national data items, and may be any length that is allowed for a national data item.

Example:

To create two accounts and add them to the collection, using the account number as the key:

```
...
WORKING-STORAGE SECTION.
01  an-account usage object reference Account.
01  a-keyed-collection usage object reference KeyedCollection.
...
Procedure Division.
...
*>      Create the Collection
INVOKE KeyedCollection "New" RETURNING a-keyed-collection.
*>      Create an Account
INVOKE Account "NewAccount" RETURNING an-account.
*>      Add the new account to the collection
INVOKE a-keyed-collection "AddKeyed"
        USING an-account, an-account::"AccountNumber"
*>      Create another Account
INVOKE Account "NewAccount" RETURNING an-account.
*>      Add the new account to the collection prior to
*>      the first account
INVOKE a-keyed-collection "AddKeyed"
        USING an-account, an-account::"AccountNumber"
```

We can now retrieve an account using the account number we specify:

```
INVOKE a-keyed-collection "ReturnKeyedObject" USING N"2" RETURNING an-
account.
```

C.1.4 Sorted collection

Sorted collections use a method within each object added to that collection to determine the actual sequence that the object references will be returned when accessed sequentially. Once an object reference has been added to the sorted collection, the character string that was associated with that object reference cannot be changed, even if the method that returned the character string would return a different value.

Unlike the other types of collections, the sorted collection cannot be created using the "New" factory method. Instead, the NewSortedCollection is used to created a sorted collection.

Example:

To create two accounts and add them to the collection, using the account holders name as the sequence order for the collection:

```
...
WORKING-STORAGE SECTION.
01  an-account usage object reference Account.
01  a-sorted-collection usage object reference sortedCollection.
...
Procedure Division.
...
*>      Create the Collection
INVOKE SortedCollection "NewSortedCollection"
        USING "GetName" RETURNING a-keyed-collection.
*>      Create an Account
INVOKE Account "NewAccount" RETURNING an-account.
*>     Set the name field
INVOKE an-account "SetName" USING "Igor"
*>      Add the new account to the collection
INVOKE a-keyed-collection "AddKeyed" USING an-account
*>      Create another Account
INVOKE Account "NewAccount" RETURNING an-account.
*>     Set the name field
INVOKE an-account "SetName" USING "Fred"
*>      Add the new account to the collection
INVOKE a-keyed-collection "AddObject" USING an-account
```

When we retrieve the object references in sequence, account number 2 (Fred) is retrieved, then account number 1 (Igor):

```
INVOKE an-ordered-collection "ReturnFirst" RETURNING an-account. *> this
                                                                 *> will return
                                                                 *> Fred’s account
INVOKE an-ordered-collection "ReturnNext" RETURNING an-account. *> this
                                                                 *> will return
                                                                 *> Igor’s account
```

© ISO/IEC 2004 – All rights reserved

C.1.5 Iterators

An iterator is an instance of the Iterator class, which contains methods for retrieving object references from a collection. Unless specified at the time the iterator is created, the order that the object references are returned through the iterator is the order that they were added to the associated collection. To retrieve object references in a different sequence, the optional sequence parameter of the CreateIterator function is used. Multiple iterators may be associated with a collection. Object references in the collection may be deleted through an iterator. Any additions to or deletions from the collection associated with an iterator, except deletions made through that iterator, invalidate the iterator. Any attempt to use an iterator method after the iterator has been invalidated will raise a CollectionException object.

When creating an iterator, sequencing may be performed by invoking a method common to all of the objects referenced in the collection. This method is referred to as the sequencing method. The sequencing method returns a national character string that is internally associated with the object references, and is used to sort them. The object references are then returned to the program in the sorted sequence.

An Iterator can be created on any collection class that inherits from the base Collection class.

Example:

Let us assume that we have created a collection and added accounts for Igor (account number 1) and Fred (account number 2):

```
...
WORKING-STORAGE SECTION.
01  an-account usage object reference Account.
01  a-collection usage object reference Collection.
01  an-iterator usage object reference Iterator.
01  another-iterator usage object reference Iterator.
...
```

We can then create an iterator sequenced by name.

```
...
*>      Create the Iterator
INVOKE a-collection "CreateIterator" USING "GetName" RETURNING an-iterator.
```

To retrieve the first account in name sequence using the iterator, we would invoke the "ReturnFirst" method of the iterator, exactly as if we were invoking the method of the same name on the base Collection class.

```
INVOKE an-iterator "ReturnFirst" RETURNING an-account. *> Returns Fred
```

This would return Fred’s account. We can then create another iterator sequenced by account number. This iterator does not interfere with the first iterator we created

```
...
*>      Create the Iterator
INVOKE a-collection "CreateIterator"
        USING "AccountNumber" RETURNING another-iterator.
```

Retrieving the first account using this iterator would return Igor's account, since Igor was account number 1.

```
INVOKE another-iterator "ReturnFirst" RETURNING an-account. *> Returns Igor
```

This iterator does not interfere with the first iterator we created. If we read the next object reference from the first iterator, it would return Igor's account, since that would be the next in sequence by name.

```
INVOKE an-iterator "ReturnNext" RETURNING an-account. *> Returns Fred
```

We can continue to use these iterators as long as we do not add or delete any object references from a-collection. If we do, then attempting to invoke any methods of `an-iterator` or of `another-iterator` will produce an error.

© ISO/IEC 2004 – All rights reserved

Bibliography

[1] ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*, 2001, 4th Edition.

[2] Merriam-Webster's Collegiate® Dictionary, Tenth Edition; Merriam-Webster, Incorporated, 2001, ISBN 0-87779-709-9.

© ISO/IEC 2004 – All rights reserved