TECHNICAL REPORT

ISO/IEC TR 19755

First edition
2003-12-01

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

Technologies de l'information — Langages de programmation, leurs environnements et interfaces logiciel système — Finalisation d'objet pour le langage de programmation COBOL

Reference number
ISO/IEC TR 19755:2003(E)

Contents

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 19755:2003, 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*.

This document is being issued in the Technical Report (type 2) series of publications (according to the Procedures for the technical work of ISO/IEC JTC 1) as a "prospective standard for provisional application" in the field of object finalization in COBOL because there is an urgent need for guidance on how standards in this field should be used to meet an identified need.

This document is not to be regarded as an "International Standard". It is proposed for provisional application so that information and experience of its use in practice may be gathered. Comments on the content of this document should be sent to the ISO Central Secretariat.

A review of this Technical Report (type 2) will be carried out not later than three years after its publication with the options of: extension for another three years; conversion into an International Standard; or withdrawal.

Introduction

This Technical Report specifies a feature for finalizing objects in COBOL. The feature is considered to be immature and not ready for standardization. The decision was made to publish the specification in a Type 2 Technical Report so that implementations can be undertaken on an experimental basis. The experience gained is expected to result in an improved specification that can progress to standardization.

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 feature for finalizing objects.

The purpose of object finalization is to free resources that will not otherwise be freed by the normal garbage collection process. Examples include files that are open, temporary work files, database connections, TCP/IP socket interfaces, and network connections.

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

1 Scope

This Technical Report specifies the syntax and semantics for object finalization in COBOL. The purpose of this Technical Report is to promote a high degree of portability in implementations of object finalization, 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 to this Technical Report

Conformance to this 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 Technical Report.

4 Terms and definitions

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

4.1
auto-method
A special kind of method that is invoked only by the runtime system. An auto-method has the characteristics of a method with certain specified exceptions.

4.2
finalizer
A kind of auto-method that is invoked by the runtime system before the resources of an instance object are reclaimed by garbage collection.

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.

6.1 Changes to 7, Compiler directing facility

[a] Add the following sentence to the end of general rule 1) of 7.2.17.3, General rules:

Automatic propagation of exception conditions from a finalizer is always disabled.

6.2 Changes to 8, Language fundamentals

[a] Add a new reserved word to 8.9, Reserved words:

AUTO-METHOD

[b] Add a new context-sensitive word to 8.10, Context-sensitive words:

FINALIZER	AUTO-METHOD paragraph

6.3 Changes to 9, I-O, objects, and user-defined functions

[a] Replace 9.3.14.1, Life cycle for factory objects, which says:

"A factory object is created before it is first referenced by a run unit.

A factory object is destroyed after it is last referenced by a run unit."

with

"A factory object is created before it is first referenced by a run unit. The state of a created factory object is reachable.

A factory object is destroyed after it has been placed in unreachable state."

[b] Replace 9.3.14.2, Life cycle for instance objects, which says:

"An instance object is created as the result of the NEW method being invoked on a factory object.

An instance object is destroyed either when it is determined that the object cannot take part in the continued execution of the run unit, or when the run unit terminates, whichever occurs first.

The timing of and algorithm for the mechanism that determines whether or not an instance object can take part in the continued execution of the run unit is implementor defined.

NOTE The process of determining whether or not an instance object can take part in continued execution and reclaiming resources unique to the object is generally referred to as garbage collection."

with

"An instance object is created as the result of the NEW method being invoked on a factory object. The object returned is in reachable state. An object created in the range of finalization immediately transitions to finalizable state.

An instance object is destroyed after it has been placed in unreachable state. The resources allocated for the object may be reclaimed when it is destroyed.

The timing of and algorithm for the reclamation of resources are implementor defined.

NOTE The process of reclaiming resources unique to the object is generally referred to as garbage collection."

[c] Insert 9.3.14.3, Object finalization, and 9.3.14.4, State transitions of objects, after 9.3.14.2, Life cycle for instance objects:

9.3.14.3 Object finalization

An auto-method, finalizer, of an object is invoked by the runtime system before the object is destroyed and the resources of an instance object are reclaimed. The execution of finalizers on an object is called finalization of the object. The range of execution of finalization includes all statements executed from the invocation of a finalizer auto-method until its termination.

NOTE This may include the execution of statements in runtime elements outside the finalizer auto-method.

If more than one finalizer is defined in a class inheritance hierarchy of an object, each finalizer shall be invoked after all finalizers that are defined in subclasses within that hierarchy have been invoked. There may be more than one possible order of invocation. The order in which the finalizers are invoked is undefined, except that no finalizer of a given class is invoked until finalizers of all its subclasses have completed. Each finalizer associated with an object shall be invoked at most once for that object. During execution of a finalizer, no other finalizers shall be invoked.

The finalization of an object is completed when control is returned to the runtime system after all associated finalizers have been executed on the object. During finalization of an object, no other objects shall be under finalization. The order of finalization among the objects is undefined.

During execution of the run unit, the timing of finalizer invocation is undefined.

NOTE Finalizer invocations may cause side effects within the COBOL run unit.

9.3.14.4 State transitions of objects

The object life cycle consists of four states: reachable, finalizable, finalized, and unreachable. The state transitions occur in the following order:

1) Reachable

   An object is reachable when it has been created in the reachable state and has not yet been determined as finalizable. Reachable objects may be referenced during the finalization of finalizable objects.

2) Finalizable

   An object is finalizable when it is determined that it can no longer participate in the continued execution of the run unit outside of finalization. The timing of and algorithm for recognizing finalizable objects is implementor-defined and may be affected by invoking the InvokeFinalizers method defined in the standard class BASE. If no finalizer is associated with an object in this state, the state of the object transitions to the finalized state. If one or more finalizers are associated with an object in this state, the finalizers are invoked by the runtime system on the object. Finalizable objects may be referenced during the finalization of other finalizable objects.

3) Finalized

   All finalizers associated with the object have been invoked and completed. One or more finalizable objects reference this object directly or indirectly through one or more finalized objects. Finalized objects may be referenced during the finalization of finalizable objects.

4) Unreachable

   The object has been finalized and no other finalizable object references this object directly or indirectly. This state indicates that the object may be destroyed and its resources reclaimed by garbage collection. The implementor shall define the time at which an object in unreachable state is destroyed.

NOTE An object reference to a finalizable object or a finalized object is not allowed to be implicitly or explicitly assigned to a data item defined in a runtime element that is not part of finalization. Therefore, neither finalizable objects nor finalized objects can become reachable again.

The state transitions that occur at the time a run unit is terminating are described in 14.5.10, Run unit termination, and 14.5.11, Abnormal run unit termination.

6.4 Changes to 10, Structured compilation group

[a] Add a new auto-method format to 10.5.1, General format:

where auto-method-definition is:

```
[ IDENTIFICATION DIVISION. ]
AUTO-METHOD. FINALIZER.
[ options-paragraph ]
[ environment-division ]
[ data-division ]
[ procedure-division ]
END AUTO-METHOD.
```

[b] Add the new end marker END AUTO-METHOD to the end markers specified in 10.6.1, End markers, General format.

6.5 Changes to 11, Identification division

[a] Add the new auto-method-paragraph to 11.1.1, General format.

[b] Add the following auto-method-paragraph specification to 11, Identification division:

11.6a AUTO-METHOD paragraph

The AUTO-METHOD paragraph indicates that this identification division is introducing an auto-method definition.

11.6a.1 General format

Format 1 (finalizer):

```
AUTO-METHOD. FINALIZER.
```

11.6a.2 Syntax rules

1) The finalizer auto-method may be specified only in the object definition of a class definition.

2) The finalizer auto-method may be specified only once in a class definition.

11.6a.3 General rules

1) An auto-method has the characteristics of a method with the following exceptions:

   [a] An auto-method may be invoked only by the runtime system.

   [b] An auto-method definition is not included in the interface of the object.

   [c] An auto-method definition is not inherited from a superclass.

   NOTE More than one auto-method definition may be specified in a class inheritance hierarchy.

2) The FINALIZER clause specifies that this auto-method definition is a finalizer.

3) A finalizer for an object shall be invoked as described in 9.3.14.3, Object finalization, and in 9.3.14.4, State transitions of objects.

6.6 Changes to 14, Procedure division

[a] Add a new syntax rule to 14.1.2, Syntax rules, under FORMATS 1 AND 2:

1a) The USING, RETURNING, and RAISING phrases shall not be specified in an auto-method definition.

[b] Change the 7th paragraph under 14.5.3, Explicit and implicit transfers of control, in part to read:

"... the execution of an EXIT AUTO-METHOD, EXIT FUNCTION, ..."

[c] Replace item 4) under 14.5.10, Normal run unit termination, which says:

“4) All instance objects are destroyed.

NOTE Any open files in an object are closed before the object is deleted.”

with

"4) All reachable instance objects are placed into finalizable state and all instance objects are finalized and become unreachable. Then all factory objects become unreachable. If the finalization in this rule causes abnormal termination of the run unit, the rules for abnormal run unit termination apply.

4a) All objects are destroyed and the resources allocated for objects is reclaimed.

NOTE Any open files in an object are closed before the object is deleted.”

[d] Replace 14.5.11, Abnormal run unit termination, which says:

“When abnormal run unit termination occurs, the runtime system attempts to perform the operations of normal termination as specified in 14.5.10, Normal run unit termination. The circumstances of abnormal termination may be such that execution of some or all of these operations is not possible. The runtime system performs all operations that are possible.

The operating system shall indicate an abnormal termination of the run unit if such a capability exists within the operating system.”

with

“When abnormal run unit termination occurs, all objects are placed in the unreachable state and finalizers are not invoked. The runtime system then attempts to perform the operations of normal termination as specified in 14.5.10, Normal run unit termination. The circumstances of abnormal termination may be such that execution of some or all of these operations is not possible. The runtime system performs all operations that are possible.

The operating system shall indicate an abnormal termination of the run unit if such a capability exists within the operating system.”

[e] Add EXIT AUTO-METHOD to the list of items that result in normal completion of a declarative procedure in 14.5.12.1.1, Normal completion of a declarative procedure, item 1.

[f] Add the following new exception to Table 14, Exception-names and exception conditions, under 14.5.12.1.5, Exception-names and exception conditions:

EC-OO-FINALIZABLE	Fatal	Invalid assignment of a finalizable or finalized object reference.

[g] Add the following to 14.8.13, Exit statement:

The EXIT AUTO-METHOD statement marks the logical end of the execution of an auto-method.

[h] Add the format for EXIT AUTO-METHOD to 14.8.13.1, General format of the EXIT statement as follows:

Format 3a (auto-method):

<u>EXIT</u> <u>AUTO-METHOD</u>

[i] Add the following new syntax rule to 14.8.13.2, Syntax rules of the EXIT statement:

FORMAT 3a

8a) An EXIT AUTO-METHOD statement may be specified only in the procedure division of an auto-method.

[j] Add the following new general rule to 14.8.13.3, General rules for the EXIT statement:

FORMAT 3a

7a) The execution of an EXIT AUTO-METHOD statement causes the executing auto-method to terminate and control to return to the runtime system.

[k] Add the following new general rule to 14.8.24.3, General rules for the MOVE statement:

2a) If the sending operand is a group item that contains one or more object references and if at least one of them references an object in the finalizable or finalized state, the receiving operand shall not be described with the BASED clause and shall be one of the following:

— a data item defined in a finalizer definition,

— an implicitly allocated record for passing a parameter during activation of a runtime element,

— a data item defined in a local-storage section,

— a data item defined in the instance definition of an instance object that is in the finalizable or finalized state.

Otherwise, the EC-OO-FINALIZABLE exception condition is set to exist.

NOTE This general rule prevents an object in finalizable or finalized state from participating in the continued execution of the run unit outside the range of finalizer execution, thus precluding object resurrection.

[l] Add the following new general rule to 14.8.35.3, General rules of format 5 of the SET statement:

9a) If the object identified by identifier-4 is in the finalizable or finalized state, the data item identified by identifier-3 shall not be described with the BASED clause and shall be one of the following:

— a data item defined in a finalizer definition,

— a data item defined in a local-storage section,

— an implicitly allocated record for passing a parameter during activation of a runtime element,

— a data item defined in the instance definition of an instance object that is in the finalizable or finalized state.

Otherwise, the EC-OO-FINALIZABLE exception condition is set to exist.

NOTE This general rule prevents an object in finalizable or finalized state from participating in the continued execution of the run unit outside the range of finalizer execution, thus precluding object resurrection.

[m] Replace general rule 1) under 14.8.38.3, STOP statement, General rules, which says:

“1) The operations described in 14.5.10, Normal run unit termination, are performed.”

with

"1) If the STOP statement is executed within the range of finalization, the execution of the run unit terminates abnormally as specified in 14.5.11, Abnormal run unit termination. Otherwise, the execution of the run unit terminates normally and the operations described in 14.5.10, Normal run unit termination, are performed.”

6.7 Changes to 16, Standard classes

[a] Add the InvokeFinalizers method to the BaseFactoryInterface specified in 16.1, BASE class:

```
Method-id. InvokeFinalizers.
Procedure division.
End method InvokeFinalizers.
```

16.1.3 INVOKEFINALIZERS method

The InvokeFinalizers method is a factory method that triggers the finalization of the instance objects that can no longer participate in the continued execution of the run unit outside of finalization. Resources held in those objects are made available for reclamation.

NOTE This method can be invoked by specifying in an INVOKE statement a class-name of an arbitrary class that inherits the BASE class directly or indirectly. The specified class-name is irrelevant to the behavior of this method.

16.1.3.1 General rules

1) The InvokeFinalizers method shall be implemented as if it were defined with a FINAL clause.

2) When the InvokeFinalizers method is invoked, the following occurs in the order specified:

a) The set of reachable objects, the set of finalizable objects, and the set of finalized objects are determined.

b) Finalization is performed on the objects in that set of finalizable objects. All objects that were found to be in either of these sets of finalizable or finalized objects become unreachable.

Annex A
(normative)

Language element lists

A.1 Implementor-defined element list

The following is a list of the language elements within this 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) States of an object (timing of and algorithm for determining finalizable objects and when to destroy unreachable objects). This item is required. This item need not be documented in the implementor's user documentation. (9.3.14.4, State transitions of objects)

A.2 Undefined language element list

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

1) Order of finalization among objects. The order of finalization among the objects is undefined. (9.3.14.3, Object finalization)

2) Order of finalizer invocation within an object. The choice of possible variations of the finalizer invocation order within an object is undefined. (9.3.14.3, Object finalization)

3) Timing for finalizer invocation. During execution of the run unit, the timing for finalizer invocation is undefined. (9.3.14.3, Object finalization)

Annex B
(informative)

Unresolved technical issues

B.1 General

The following technical issue has proven difficult to resolve and has not been addressed in the finalizer design. This issue is presented here in order to obtain feedback from reviewers and early implementors.

B.2 Requirement for garbage collection timing

The INVOKEFINALIZERS method makes objects available for reclamation by completing all the steps necessary to place them in the unreachable state. A programmer might expect the reclamation of resources to be complete upon return from this method. The reclamation of resources is the job of garbage collection. If this technical report were to specify that garbage collection did occur prior to the return from the method, it would be an exception to the specification that timing and algorithm of garbage collection is implementor defined. Forcing actual resource reclamation (garbage collection) as a part of INVOKEFINALIZERS might result in undesirable system performance.

Annex C
(informative)

Concepts

C.1 Finalization in COBOL

Finalization is a process that may be defined for an object for the purpose of cleaning up and releasing resources before garbage collection.

C.1.1 Object life cycle

The object life cycle in COBOL consists of 4 states:

— Reachable

— Finalizable

— Finalized

— Unreachable

In COBOL there are two types of objects – factory objects and instance objects. When subject to normal run unit processing, an object is created in the reachable state. Factory objects are created at their first reference in the run unit and persist until the termination of the run unit. Instance objects are created using the factory method “new” and exist until they are in the unreachable state.

Objects created by processes running under finalization – that is, created by run time elements that are running under the control of a COBOL finalizer are always created in the Finalizable state. No objects created or referenced while under finalization may be placed or exist in a reachable state. No objects ever transition from Finalizable or Finalized to Reachable.

Objects that have completed their finalizers are in the finalized state. Once no references to these objects exist in any finalizable object, their state becomes unreachable and their resources are available for reclamation via garbage collection.

Factory objects are not placed in the unreachable state until run unit termination.

C.1.2 Object resurrection

Object resurrection is a process of restoring finalizable or finalized objects to a reachable state. In COBOL, the state of a finalizable or finalized object is never restored to reachable. An object reference for a finalizable or finalized object can be referenced within the range of finalizer execution, but not outside that range, and the object’s methods can be executed. Rules of the MOVE and SET statements prohibit the moving or setting of those object references into storage locations that are accessible outside the range of finalizer execution. As a result, returning finalizable or finalized object references from a runtime element is not allowed, either using a RETURNING phrase or using the BY REFERENCE phrase of a CALL or INVOKE statement. Thus participation in the continued execution of the run unit outside the range of finalizer execution by an object in finalizable or finalized state is precluded.