ISO/IEC JTC 1/SC 22/WG4 N 0163

Date: 2002-05-21

Reference number of document: **WDTR 19755**

Version 1.1

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

Secretariat: ANSI

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

Document type: Technical report
Document subtype: n/a
Document stage: (20) Preparation
Document language: E

Acknowledgement notice

COBOL originated in 1959 as a common business oriented language developed by the Conference on Data Systems Languages (CODASYL). The authors and copyright holders of the original copyrighted material specifically authorized the use of the material, in whole or in part, in COBOL specifications. That authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications. CODASYL requested that the following acknowledgement be placed in the preface to any such publication.

Acknowledgment

Any organization interested in reproducing the COBOL standard and specifications in whole or in part, using ideas from this document as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication:

> COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.
>
> No warranty, expressed or implied, is made by any contributor or by the CODASYL COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.
>
> The authors and copyright holders of the copyrighted materials used herein
>
> > FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell
>
> have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

Any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgment of the source.

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.

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 19755 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 19755 extends the COBOL specification defined in ISO/IEC FDIS 1989:2002, Information technology — Programming languages, their environments and system software interfaces — Programming language COBOL. It adds a feature for finalization of objects in Programming language COBOL.

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

Introduction

This proposed draft 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, 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 proposed draft Technical Report specifies the syntax and semantics for object finalization in COBOL. The purpose of this proposed draft 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 FDIS 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 19755. 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 19755 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 FDIS 1989:2002, *Information technology — Programming languages, their environments and system software interfaces — Programming language COBOL.*

3 Conformance to this proposed draft Technical Report

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

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 FDIS 1989:2002.

6 Changes to ISO/IEC FDIS 1989:2002

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

6.1 Changes to 7, Compiler Directives

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

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, 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. It is undefined in which order the finalizers are invoked, except that the finalizers of each subclass will be invoked before the finalizer of a superclass. 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 will 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:

[<u>IDENTIFICATION</u> <u>DIVISION</u>.]
<u>AUTO-METHOD</u>. <u>FINALIZER</u>.
[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, General format, under End markers.

6.5 Changes to 11, Identification Division

[a] Add the new auto-method-paragraph to 11.1, Identification division, 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 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 Procedure division, 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] Replace the paragraph in 14.5.3, Explicit and implicit transfers of control, which says:

"There is no next executable statement when the execution of an EXIT FUNCTION, EXIT METHOD, EXIT PROGRAM, GOBACK, or STOP statement transfers control outside the COBOL source element."

with

"There is no next executable statement when the execution of an EXIT AUTO-METHOD, EXIT FUNCTION, EXIT METHOD, EXIT PROGRAM, GOBACK, or STOP statement transfers control outside the COBOL source element."

[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.

[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

An EXIT AUTO-METHOD statement shall be specified only in an auto-method procedure division.

[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:

3) 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 allocated record for passing a parameter,

— 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.

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

11) If the object identified by identifier-5 is in the finalizable or finalized state, the data item identified by identifier-4 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 allocated record for passing a parameter,

— 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.

[m] Replace general rule 1 of the Stop statement under 14.8.38.3, 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 to be destroyed and makes the resources held in those objects available for reclamation via garbage collection.

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 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) States of an object (timing of and algorithm for determining finalizable objects and when to destroy unreachable objects). This item is required. This item shall 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 proposed draft 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

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

B.2 Whether Finalizers should run under abnormal termination

Even under abnormal termination, some might want some finalizers to run. Under the specification of this proposed draft Technical Report, all objects are put in unreachable state and no finalizers are run. The reason for this is that it may be impossible or inappropriate to run certain finalizers under abnormal termination. Users may want a mechanism to determine which finalizers may be run under abnormal termination.

The factory finalizers are more useful if this feature is implemented.

B.3 Finalizers in factory object

Finalizers in a factory object might be useful. This proposed draft Technical Report does not provide a facility to specify a finalizer in the factory definition of a class. This is because the life cycle of factory objects might not be the same among the implementations. One implementation may instantiate all factory objects at the beginning of the run unit, another may instantiate a factory object only when it is used for invocation of its method. Because of this, some of the finalizers might not run at the termination of run unit. This presents a portability issue.

B.4 Returning finalizable or finalized object references

It is intended that the resurrection-related checks be done when a MOVE or SET statement is executed. As a result, returning finalizable or finalized object references from an activated runtime element is not allowed, either using the RETURNING phrase or using the BY REFERENCE phrase in the USING phrase.

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.2 Auto-methods

An auto-method is a method in every respect with the following exceptions:

— An auto-method is not included in the interface of an object

— An auto-method definition is not inherited from a superclass

— An auto-method definition in a subclass does not override an auto-method definition in a superclass

— More than one auto-method of the same type may exist in the class hierarchy of an object

— An auto-method does not support the following:
  – the RAISING, RETURNING, and USING phrases in the procedure division header
  – the RAISING phrase of an EXIT statement or a GOBACK statement

C.2.1 FINALIZER auto-method

The auto-method FINALIZER establishes a finalizer for the class in which it is declared. Finalizers are invoked in an implementor-defined order against objects that are in finalizable state. Within a run unit, finalizers run “single-threaded”; that is, only a single finalizer may be active at any one time.

C.3 Finalizer references to other objects

Objects created during the execution of a finalizer are created in the finalizable state. A finalizer may reference any object that is in the reachable, finalizable, or finalized state. No object may transition from a finalized or finalizable state to the reachable state.

C.4 Order of Finalizer invocation

The order of finalizer invocation is implementor defined at the object level. That is, the implementor defines the order in which objects have their finalizers invoked. However, within a specific object, the finalize auto-method is invoked for each subclass before the finalizer in the superclass is invoked. Each class in the inheritence hierarchy has its finalizer (if any) invoked in turn until all finalizers for that class have been invoked. Where multiple possible finalizers may be executed, the implementor defines the order of such execution.

C.5 InvokeFinalizers

InvokeFinalizers is a method of the base class. It triggers the runtime system to check for objects that are in the finalizable state. Once this set of objects is determined, the finalizers of these objects are invoked.

C.6 Avoiding side effects

No restrictions are placed on the Finalizer auto-method as to programs or functions that may be called or methods that may be invoked. Because of this, programmers should be aware of the possibility that side effects might occur within the normal or finalizer execution paths of the run unit.

The following programming practices can mitigate these side effects:

— Use file and record locking facilities, if provided in the COBOL implementation.

— Do not use external data and external files in a program, function, or method that might be used within the range of finalization.

— Use the recursive attribute for programs that are to be called within the range of finalization. Use local-storage instead of working-storage in such programs.

— A non-recursive program called within the range of finalization might already be in the active state. If this occurs, the fatal EC-PROGRAM-RECURSIVE-CALL exception condition is set to exist. Termination of the run unit can be avoided in such cases by making use of declaratives in the calling program, function, or method. Execution can continue by coding a RESUME statement within each declarative.

Bibliography

[1] ISO/IEC Directives, Part 3, *Rules for the structure and drafting of International Standards*, 1997.

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