COBOL 2002 –
The Good, the Bad, and the UGLY

Session 8204

August 25, 2005 – 11:00 a.m.

Speaker: Tom Ross (IBM) – based on an original presentation created by:
William M. Klein (WMK)
wmklein@ix.netcom.com

Revised: Tuesday, August 23, 2005

NOTE:

This (lengthy) document was originally created for the Summer SHARE in New York, NY in 2004. It has been updated for the (51st Annual) SHARE in Boston, MA in August 2005.

Additional (ongoing) changes may be done in the future (between SHARE meetings). To see the most recent version (in HTML format), check at:

http://home.comcast.net/~wmklein/SHARE/S8204.htm

Table of Samples and Tables

Introduction

So far, IBM has neither rejected the idea of providing a "fully conforming" ISO 2002 compiler, nor have they made any public (or semi-privately to the SHARE LNGC project) commitment to do so. Their current response of "RECOGNIZE" against the SHARE requirement for a fully conforming compiler has the official meaning of:

> "IBM agrees with the request and a solution ***appears to be a desirable objective***. A solution, however, may ***not*** presently appear feasible or implementable. No IBM commitment is made or implied as to the eventual delivery of an acceptable solution. "

Even if there were not issues of priorities and development resources (for IBM) and the ultimate cost to its customers, there are some features that existing Enterprise COBOL (and possibly other IBM customers of other compilers) might find either undesirable or unnecessary. This session will attempt to *BRIEFLY* present both the current status of IBM's implementation of 2002 Standard features and to raise some "flags" about some features that current customers might question for ultimate implementation and purchase.

The appendices of this document include a detailed analysis of which "substantive changes" have and have not already been implemented, along with a brief introduction to three of the more complex features of new Standard.

NOTE: This sessions was originally created by Bill Klein and reflects his input and options; not (necessarily) those of Tom Ross, IBM, or any other group, organization, or company..

Who is Bill Klein and what does he have to do with this and other Standards?

I have:

- Worked on the J4 (formerly X3J4) COBOL Committee (initially representing a compiler vendor, then representing myself as an individual)
- Worked on the CODASYL COBOL Committee (when it still existed)
- Worked for an IBM partner on the design (and "long-dead") implementation of SAA COBOL L2 (and the never seen L3)
- Worked with GUIDE (and now SHARE) to help them understand what has, is, and might be happening with ANSI and ISO COBOL Standards

As far as ISO 1989:2002 (the ISO 2002 COBOL Standard) goes:

- I worked on it during the first two-thirds (or so) of its development
- I really liked (and still like) many if not all of the new and enhanced features
- By the time of the final public review period, I came to believe (and expressed myself rather vocally) that the final product was ***NOT*** good for the COBOL programming community (and whether or not it is good for compiler vendors may or may not depend upon whether a fully conforming compiler ever becomes available)
- To this day, I don't know whether I think it will or won't benefit COBOL programmers, vendors, or others.
- I don't really know if *ANY* vendor will ever produce a "fully conforming" 2002 conforming compiler – much less if any programmer will ever use such a product when/if it becomes available. (I certainly doubt that there will ever be any "certification" tests for it – so that "users" can TELL if a compiler is or is not truly conforming.)

What is the status of the COBOL Standard?

- The ISO 2002 Standard was "approved" in December of 2002
- The '85 Standard (ANSI and ISO) – with or without its two Amendments - is no longer an officially recognized Standard
- The first TR (Technical Report) for "Object Finalization" has been completed and approved
- Two more TR's (for native XML support and for Collection Classes) are progressing and should be finished and approved within the next year or two (or so)
- The ***NEXT*** ISO COBOL Standard is currently scheduled to be finished and approved in 2009

  For details of the ongoing (and future) revision work, see

  http://www.cobolportal.com/j4/files/05-0108.doc

- The first draft of this next Standard should be available for "public review" in the relatively near future.
- There is (as far as I know) currently no FIPS (US government) COBOL (or other programming language (except maybe ADA and SQL) Standard – much less government recognized validation suite for COBOL. If any vendor *claims* to conform to the ISO 2002 COBOL Standard, then you have only their word for it.

Processor-dependent language elements

In the past, there has been some confusion about certain items (for example Accept/Display with "Screen Section" and whether or not these items must be implemented in a "fully conforming" ISO 2002 COBOL compiler. Before looking at specific changes in the new Standard, understanding the "processor-dependent" terminology is important.

NOTE:

The following text comes directly from the ISO 2002 Standard, page 4, in the section,

3.1.5 Processor-dependent language elements

"Processor refers to the entire computing system that is used to translate compilation groups and execute run units, consisting of both hardware and relevant software. Language elements that depend on specific devices or on a specific processor capability, functionality, or architecture are listed in B.3, Processor-dependent language element list. To meet the requirements of standard COBOL, the implementor shall document the processor-dependent language elements for which the implementation claims support. Language elements that pertain to specific processor-dependent elements for which support is not claimed need not be implemented. The decision of whether to claim support for a processor-dependent language element is within an implementor's discretion.

Factors that may be considered include, but are not limited to, hardware capability, software capability, and market positioning of the processor.

When standard-conforming support is claimed for a specific processor-dependent language element, all associated syntax and functionality required for that language element shall be implemented; when a subset of the syntax or functionality is implemented, that subset shall be identified as a standard extension in the implementor's user documentation. The absence of processor-dependent elements from an implementation shall be specified in the implementor's user documentation.

An implementation shall provide a warning mechanism at compile time to indicate use of syntactically-detectable processor-dependent language elements not supported by that implementation. Although this warning mechanism is required to identify processor-dependent elements that are not supported, it is not required to diagnose syntax errors within this unsupported syntax. The implementor is not required to produce executable code when unsupported processor-dependent language elements are used.

* * * * * * * * * * * * * *

Therefore, IBM could provide a "fully conforming" compiler with as many (or as few) of these items as they deem "appropriate." Such items are marked as

"(PD)"

In the appendices detailing substantive changes. And IBM's current implementation status.

Summary IBM's implementation of "Substantive Changes"

In the ISO 2002 COBOL Standard, there are two annexes describing "Substantive Changes".

- One deals with those "potentially affecting" existing ('85 Standard conforming) programs

  And

- One dealing with those "not affecting existing programs".

There are 40-plus of the first type and 100-plus of the second type. However, it is important to understand what constitutes a "single entry" in these lists. For example, both

- Full Object Orientation

  And

- Making the word "OF" optional in certain Report Writer syntax

each count as a "single entry". Therefore, when reading the material in this handout, a certain level of caution is required. On the other hand, some general perceptions can be taken from what has and has not already been implemented by IBM.

For a (free) PDF copy of the entire "Substantive changes not affecting existing programs" Annex to the '02 Standard, see:

http://www.cobolstandards.com/annexf.pdf

NOTE: this count has not been updated since the latest requirements were submitted to IBM and since IBM submitted their latest responses.

Already Implemented – 56

- Of which 23 are implemented, not by Enterprise COBOL, but by the Report Writer add-on product
- For details, see:

  Appendix A.1: Items currently Available/Implemented in IBM Enterprise COBOL V3

Comparable Functionality is available in Enterprise COBOL – 6

- SHARE has already submitted 3 requirement against these
- For details, see:

  Appendix A.2: Items with comparable functionality in IBM Enterprise COBOL V3R3

Items Partially implemented in Enterprise COBOL - 5

- SHARE has already submitted 1 requirement against these
- For details, see:

  Appendix A.3: Items Partially Implemented in IBM Enterprise COBOL V3R3

Items not yet implemented in Enterprise COBOL – 88

- SHARE has already submitted requirements against 27 of these items
- Of these, 17 items are in the “processor dependent” list (and need not be implemented by a fully conforming compiler – see above)
- For details, see:

  Appendix A4.: Items not yet implemented by IBM

NOTE:

The specific features implemented (fully and partially) are slightly different for IBM’s COBOL compilers for VSE, VM, OS/400, and even Windows and AIX. This presentation deals only with the features available for the current z/OS compiler.

IBM's implementation of "Substantive Changes" – What's New

With Enterprise COBOL V3R4, IBM introduced extended support for National (Unicode compatible UTF-16) Data types – implemented in accordance with the specification in ISO 2002 COBOL. As indicated in the LRM "Summary of Changes":

Support for national (Unicode UTF-16) data has been enhanced. Several additional kinds of data items can now be described implicitly or explicitly as USAGE NATIONAL:

- External decimal (national decimal) items
- External floating-point (national floating-point) items
- Numeric-edited items
- National-edited items
- Group (national group) items, supported by the GROUP-USAGE NATIONAL clause

Some detailed issues (good and bad)

The following is presented as ***MY*** opinions only - and (hopefully) it is clear that it is my "best guess" as to things that IBM customers may and may *NOT* want IBM to spend "development resources on".

There are definitely items listed below that I think some customers will think are the “best thing” since INITIALIZE while others will think are the worst thing since NESTED PROGRAMS.

I have only included some of the changes/enhancements on the “extreme” (IMHO) side of desirability and undesirability. There are many other changes (listed in the appendices) that specific customers will either like or dislike.

When reading these opinions and descriptions, remember that unless otherwise specifically indicated, there is nothing in the ISO 2002 Standard that will stop existing and new programs from continuing to be written and developed so that they “look like” ***AND*** behave like ’85 Standard programs. However, many of us know what happens if the compiler “accepts” new features – whether or not a shop WANTS those new features to be used.

Reference format changes

Possibly the most visible area of change in the 2002 Standard, is how programs appear and are stored in source format. It is interesting to note that many (most?) of the changes listed below (both good and bad) are already available in many other COBOL compilers. It is my impression that users of these features in other environments “love them”; it is equally my impression that many IBM mainframe customers will ***HATE*** them!

GOOD: In-Line Comments

For example:

Sample 1 – Inline Comments

```
05  Field-X Pic XX      *> Used in calculating the current THINGY
          …
MOVE ABC    to XYZ   *>  Current-XYZ
                LMN  *>  Saved XYZ
```

Notice that an apostrophe rather than a quotation mark may now be used in a Standard conforming program – not just as an IBM extension. (The QUOTE figurative constant retains it meaning of ***ALWAYS*** being a quotation mark in a Standard conforming program.)

GOOD: New Style Literal continuation

For example,

Sample 2 – New Style Continuation of literals

```
Move  ‘ABC’-
          ‘XYZ’      to 6-bytes
```

WEIRD: Identification Division header is optional

Especially with OO syntax, there will be many, MANY new "nested" source units. Many of them no longer start with "Program-ID" paragraphs. As "class-id," "function-id," etc. at the beginning of a "source unit" seems to be 'self-documenting" – without "Identification Division" coded before it - this is viewed (by some) as creating "more readable" programs.

BAD: A-/B-margin removal

For (extreme) example,

Sample 3 – Removal of A-/B-Margins

```
Procedure Division. Para1. Move
ABC to
zZz. XYZ
Section.  CALL
"ABC"-
         "DEF"
```

BAD: 255 character (not byte) lines

A conforming compiler must allow 255 characters (that may be a mixture of "alphanumeric" and "national" characters) along with any required "control characters" on each source line. This may (in my opinion) be a problem for many ways of viewing (and storing) source code – not to mention any and all third party development tools.

Sample 4 – Source code lines up to 255 (mixed alphanumeric / national) characters

```
Move "ABCD杏沓枉8901234567892123456789312345678941234567895123456789612345678971234567898123456789912345678 9a123..." to Recv-Field
```

BAD: potential problems with loss of columns 73-80 information

Although "fixed format reference format" is the default, the 2002 Standard requires a compiler to also support "free form reference format". In that format, there (appears) to be no place (other than inline comments) for placing any of the information that many existing IBM mainframe sites place in columns 73-80 (e.g. revision history, responsible programmer, etc).

Similarly, any information included in columns 1-6 must be removed. My personal experience is that (other than sequence numbers) this area hasn't been used much in IBM shops. However , some vendors that distribute COBOL source code – with "fixes" identified by sequence number, will have problems with this. It will, of course, (finally?) kill off the little used BASIS, INSERT, DELETE feature that very few customers know that IBM still supports.

Performance issues

Standard arithmetic

NOTE: For an introduction to what "Standard Arithmetic" is and does, see:

Appendix E: Introduction to Standard arithmetic

This feature definitely has PROs and CONs. When this feature is turned on (and it is OFF by default), the results of many (not all) numeric operations will be more portable (and predictable) across all (ISO 2002 conforming) compilers. On the other hand, this does mean that some results will be both ***MORE ACCURATE*** and ***DIFFERENT FROM*** existing results yielded by IBM Enterprise COBOL. Some IBM shops already went thru this type of change – when moving from OS/VS COBOL to VS COBOL II Or later compilers. It was surprising (to some) that many end-users were more interested in getting the "same answer" rather than a "more accurate answer" for the same calculation.

However, it is my opinion that the performance issue of requiring intermediate results to be calculated "as if" the compiler were using a "decimal floating point" item with 31-digits of precision and 3-digits of exponent, will stop many "mission critical" applications from using this feature – even when and where available. Only the compiler (and run-time) vendors will be able to determine exactly how well they can "optimize" this feature and still yield the required results.

The following is the ISO 2002 definition of an "SIDI" or "Standard intermediate data item" that is at the heart of the "standard arithmetic" feature of the ISO 2002 Standard.

8.8.1.3.1 Standard intermediate data item

A standard intermediate data item is of the class numeric and the category numeric. It is the unique value zero or an abstract, signed, normalized decimal floating-point temporary data item. The internal representation shall be defined by the implementor. Implementors may use whatever method or methods they wish as long as the results conform to the rules for standard intermediate data items.

> NOTE The internal representation of a standard intermediate data item is permitted to vary so that implementors can choose the most efficient implementation for the circumstances.

When standard arithmetic is in effect the following rules apply:

1) Any operand of an arithmetic expression that is not already contained in a standard intermediate data item shall be converted into a standard intermediate data item.

   > NOTE This rule covers such cases as an arithmetic expression that contains only one operand and no operator. For example, IF (A = 1) and COMPUTE A = B.

2) The size error condition is raised and the EC-SIZE-OVERFLOW or EC-SIZE-UNDERFLOW exception condition is set to exist if the value is too large or too small, respectively, to be contained in a standard intermediate data item. (See 14.6.4, SIZE ERROR phrase and size error condition.)

   > NOTE Underflow is treated as a SIZE ERROR and is not rounded to zero.

8.8.1.3.1.1 Precision and allowable magnitude

A standard intermediate data item has the unique value of zero or a value whose magnitude is in the range of

10–999 (1.000 000 000 000 000 000 000 000 000 000 0E-999)

through

101000 – 10968 (9.999 999 999 999 999 999 999 999 999 999 9E+999)

inclusive, with a precision of 32 decimal digits. A standard intermediate data item shall be truncated or rounded to fewer than 32 digits only as explicitly specified.

Common Exception handling

NOTE: For an introduction to what "Common Exception Handling" is and does, see:

Appendix D: Introduction to Common exception processing

The ISO 2002 Standard introduces a "common" way of detecting (and if desired recovering from) all types of run-time exceptions and problems. Similar (in some but not all ways) to LE's condition handlers, this feature may provide significant advantages to those creating applications requiring 24x7 "always up" behavior. This feature is quite similar to both PL/I's "ON units" and CICS's EXEC CICS HANDLE CONDITION statements. Like these, however, it is

totally UNSTRUCTURED programming!!!

Logic for detecting and recovering from such exceptions is not coded "where it might happen" but in a new type of DECLARATIVE. Syntax is provided to either return control (after the exception is dealt with) to either a specific "label" or the (implicitly) next logical statement. Again, from a personal perspective, I would guess this will require HUGE amounts of object code and may cause some programs to "walk rather than run"!

NOTE:

To the best of my knowledge, IBM's LE developers have "glanced at" but not studied in detail how this feature will fit into an LE condition-handling environment. Furthermore, there are certain "exceptions" that IBM will be forced to "detect" that IBM mainframe COBOL programmers have come to expect to be handled "quietly and without error reporting".

Can we say:

"memories of NUMPROC horrors of conversion from OS/VS COBOL to later COBOLs"

Table SORT

In the past some programs avoided using the (better performing) SEARCH ALL statement, as they were working with "unsorted" tables (and the programmer didn't know how to or want to sort the table). With the '02 Standard, it is now possible to easily (as far as the COBOL source code goes) sort a table – or a level of a multi-level table. Thereby, improving the run-time performance of a (lengthy) sequential SEARCH.

Sample 5: Table SORT

```
01 group-item.
      05 tabl occurs 10 times
         indexed by ind
            ascending elem-item2
            descending elem-item1.
         10 elem-item1 pic x.
         10 elem-item2 pic x.
...
 move "l3n3m3p3o3x1x1x1x1x1" to group-item
 sort tabl
 search all tabl
      at end
          display "not found"
      when elem-item1 (ind) = "m"
          if (elem-item1 (ind – 1) = "n")
           and (elem-item1 (ind + 1) = "l")
              display "elem-item1 is descending order – 2nd key"
          else
               display "sort failed"
          end-if
  end-search.
```

Note: For those not familiar with the technique, IBM does document how to do the equivalent of a table sort – by using Input/Output procedures. See (for example),

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/igy3pg20/1.12.10.2

CALL prototypes and compile-time parameter checking

Although not strictly speaking a "performance issue," the new facility for creating "CALL prototypes should provide (potentially significant) debugging assistance and production ABEND avoidance. To me this is an important type of "performance" enhancement.

With this feature (optional for the CALL statement and "built-in" for the new ISO-conforming OO syntax), a programmer can store in an (external) repository a template or prototype for either a single subprogram or a group of programs with similar expected passed parameters. Then when any calling program is compiled, it can be checked (and verified) that the calling program is passing the correct parameters (size, category, etc) to the subprogram with the defined "CALL prototype."

Detailed "oddities" and miscellaneous

NOTE:

Some of the following I *like* but I will be mentioning them because I think some IBM sites either won't like them - or won't want to ***pay*** for them.

GOOD: Various new Standard-defined data types

The ISO 2002 Standard introduces:

- 4 sizes of “true binary” (cf. TRUNC(BIN)) data USAGEs
- 3 sizes of floating point items
- Pointer and Procedure-Pointer types (already available with Enterprise COBOL)
- Bit and Boolean data items

The new “true binary” items will (finally) allow COBOL to access and work with one-byte binary items that previously have required “odd” work-arounds.

Although the ISO Standard does NOT require that the new floating point items need to be implemented stored in IEEE floating point format, this is certainly allowed (and most implementors are using this format). If IBM were to do this, this would increase (in my opinion) inter-language operability support as well as allowing a single COBOL program to share IBM hex-format floating-point data with some other programs (and files) and IEEE format data with other programs and files.

The Boolean format is probably LESS useful that the BIT format, but is similar (not identical) to what is already available with IBM’s OS/400 COBOL (as well as many other vendor’s existing COBOL compilers). It also provides a method of storing “bit” information in external media in a human-readable format.

BAD: IF NUMERIC on BINARY items (against picture clause)

For example,

Sample 6: NUMERIC Class test on BINARY fields

```
  05 Group-1.
        10  Binary-1   Pic   S9(04) Binary.
      …
Move High-Values to Group-1
If Binary-1 Numeric
     Display “Not true for ISO 2002”
```

BAD: Lots of new reserved words

There are many, MANY, new reserved words. Enterprise COBOL does “flag” (I-level) such words, so programmers should (if they want to “protect themselves” for the future) look at such messages – and rename their procedures and fields accordingly. To the best of my knowledge, the IBM CCCA product does not (yet) detect such names.

There is also the new concept of “words reserved in context”. These should not cause programmers any problems. These work like the current intrinsic function names, i.e. they are valid as user-defined words but have specific meanings in specific syntax.

For example:

- ADDRESS
- ALIGNED
- ALLOCATE
- AS
- BIT
- COLUMNS
- CONDITION
- CONSTANT
- CURSOR
- DEFAULT
- FACTORY
- FORMAT
- FREE
- GET
- INHERITS
- LOCALE
- METHOD
- MINUS
- NESTED
- OPTIONS
- OVERRIDE
- PRESENT
- PROPERTY
- PROTOTYPE
- RAISE
- RAISING
- REPOSITORY
- RESUME
- RETRY
- RETURNING
- SCREEN
- SELF
- SHARING
- SOURCES
- SUPER
- UNIVERSAL
- UNLOCK
- VALID
- VALIDATE

WHO KNOWS: the "report writer is required - but may require add-on product" question

In the 2002 Standard, the “report writer feature” is required to be available. However, the vendor may use a “separate product” to provide this functionality (just as they may for “SORT” and even file I/O). Therefore, this probably will make little difference to IBM’s customers – either those with or without the current add-on RW product.

For a useful (interesting) paper on Report Writer and the ’02 Standard – including ***LOTS*** of examples, see:

http://www.cobolstandard.info/rw.htm

GOOD (MAYBE): ALL customers will be getting ***AND*** paying for "internationalization" support in the compiler

In today’s world, many (possibly most) IBM mainframe customers work in a multi-national and multi-cultural world. There are *SIGNIFICANT* enhancements in the ISO 2002 Standard to assist in the development and deployment of “international” applications. Certainly full Unicode (ISO 10646) support (which is almost identical to that in Enterprise COBOL) will be widely used. (Well the ’02 Standard, like Enterprise COBOL, doesn’t include “full” ISO 10646 support, as it doesn’t support “composed” characters.)

However, the ISO 2002 Standard also provides a lot of support for “LOCALE-based” run-time editing and behavior. It is unclear to me whether this (similar to traditional Unix and C solutions to this issue) feature is something that all current IBM customers will want to pay for.

NOTE:

If you are a current Enterprise COBOL customer, do you know (and understand and care about) the two existing implementations of PIC N? If you don't care about this, you may not want IBM to spend resources on the additional ISO 2002 defined internationalization features.

Check out what IBM introduced as one area of major enhancement in Enterprise COBOL V3R4 – i.e. NATIONAL numeric support

BAD, BUT: ALLOCATE and FREE and "BASED" phrase vs current IBM support for SET ADDRESS OF linkage-section-items

Many IBM sites will want to discourage the average COBOL programmer from doing their own ALLOCATEs and FREEs of storage – even if this facility is already available to them via LE callable services. It will be quite disappointing that the ISO COBOL Standard uses a very different syntax to define "structures" for which run-time addressability is established. Rather than the existing IBM approach of using Linkage Section group items, the ISO 2002 Standard allows for the "BASED" phrase in any data division section – and also provides for default pointer items.

It is certainly my best guess that IF IBM were ever to provide a fully conforming 2002 compiler, that they would continue to support the existing widely used syntax and semantics.

ALSO SCARY, BUT USEFUL: Bit twiddling

Do you want your COBOL programmers to be able to do "bit-twiddling" (without using the user-unfriendly LE callable services)?

Ready or not, now they will be able to!!!

Sample 7: Bit (and/or Boolean) manipulation

```
01 My-flag PIC 1111 USAGE BIT VALUE B"0000".
01 My-flag-2 PIC 1111 USAGE BIT.
      ...
      MOVE B"0011" to My-flag-2 *> Initialize My-flag-2
            ...
*> set the bits in My-flag to the
*> reverse of My-flag-2, 1100.
      COMPUTE My-flag = B-NOT My-flag-2
      ...
*> turn off all the bits in My-flag.
      COMPUTE My-flag = My-flag B-AND B"0000"
      ...
*> set bit 1 ON in My-flag-2, keeping other bits unchanged.
      COMPUTE My-flag-2 = My-flag-2 B-OR BX"8"
      ...
*> Alternatively, the last COMPUTE statement could be replaced by:
*> set bit 1 ON using reference*> modification

MOVE B"1" to My-flag-2(1:1)
```

DOES ANYONE CARE: full (Java and C++ "independent") OO

The ISO 2002 OO definition is COBOL-based and should (as far as I can tell) work in

- A COBOL-only OO environment
- A mixed COBOL and C++ (or C#) environment
- A mixed COBOL and Java environment

When (if) any vendor (IBM or otherwise) actually provides a *USEFUL* OO class library for the z/OS environment, then this may (or may not) become relevant to IBM's COBOL customers. Until then, their current non-Standard Java-targeted (and almost unused) support is as good as anything (IMHO).

GOOD, BUT: VALIDATE

NOTE: For an introduction to what the "VALIDATE" facility is and does, see

Appendix C: Introduction to the Validate facility

The new VALIDATE facility really meets the needs of the 1970 an 1980's COBOL programmer!

This facility provides for a data-driven way of validating full records and to take either remedial actions or to appropriately issue warnings and error messages. To the extent that new and existing programs NEED data (input) validation and remediation logic, this new approach will be useful. Unfortunately (for some – not all) customers, this is about two decades too late for such a new feature. Learning how to exploit (and debug) it may prove to be a great barrier to its widespread adoption in existing IBM mainframe shops. (It is my understanding that (like report writer), it is highly valued in those shops and environments where it is currently available and in use.

TRIVIAL, BUT: WRITE from Literal

Just as an example of some of the "small" changes that will be available with a 2002 conforming compiler, the following syntax will now be valid:

Sample 8: WRITE from "literal"

```
Write Some-Record from Spaces
Write Other-Record from All "X"
```

NICE BUT:

The following is a list of just a few of the other enhancements introduced in the ISO 2002 COBOL Standard that provide "nice" functionality – but (potentially) add additional complexity to compilers, programs, and debugging. It is hard for me to forecast which (if any) would be used by IBM mainframe customers – when/if they become available – and which would be "discouraged" by shop standards and/or lack of training and development/debugging tools.

- Numeric items may be defined up to 31 digits (Available with ARITH(EXTEND) compiler option in Enterprise COBOL – for some, but not all, data types)
- FORMAT clause can do (implicit) encoding conversions upon input or output of file I/O (not just for ASCII tape files)
- Implementor defined successful file status codes (cf. "FS=97" for VSAM file that could now be "FS=0x")

- Inline PERFORMs may have one or more AFTER phrase
- Non-floating currency symbol can appear at the end of a PICTURE clause
- READ PREVIOUS and START LESS THAN (Also START FIRST, START LAST)
- Recursion and Local-Storage section (in both OO ***and*** procedural code)
- REPLACE ALSO statement now may be “cumulative” rather than simply canceling and replacing previous REPLACE statements
- SET condition-name TO FALSE (and WHEN FALSE clause)
- “Split” RECORD and ALTERNATE RECORD keys (keys made up of multiple non-contiguous fields)
- STOP WITH STATUS (possible replacement for either RETURN-CODE or CEE3ABD)
- “Strong typing” (via TYPEDEF and TYPE attribute) Similar but not the identical to the new SAME clause (cf. COBOL/400 “LIKE” clause)
- Subscripting using arithmetic expressions
- Underscore in user-defined words (***NOT*** treated as equivalent to a hyphen)
- User-defined functions

New (and better?) Ways of doing **Old** *tasks*

This section provides some examples (not exhaustive) of some of the other specific features of the ISO 2002 COBOL specification that I (personally) think that some IBM customers would like – even if they aren't "world shaking" enhancements as most of this can be done (in one way or another) today.

Accept, Display, and Screen Section

This document will ***NOT*** go into the details of this feature. However, it is worth noting that ***IF*** IBM were to see sufficient business need in today's environment (not what would have been "nice" in 1970) that this feature could easily be used to create "portable" source code for IMS/DC, CICS, and/or ISPF applications (that have "character based" screen I/O). Furthermore, this would provide an easy migration solution to and from various existing COBOL Unix applications.

As this is a "processor dependent" feature in the 2002 Standard, a fully conforming IBM compiler need not provide this facility and personally, I would be surprised if IBM were to include it in a future z/OS compiler.

Concatenation Expressions

You can now make a single literal out of two. Possibly the most common usage of this can be accomplished by the IBM extension of z-literals. However, the facility is a "general" one in the ISO 2002 Standard.

Sample 9: Concatenation Expressions

```
Process Apost
   ...
If  "This is a null-terminated String" & X"00"
 = z"This is a null-terminated String"
      Continue
Else
  Move "IBM" & " Extension" & " isn" & quote & "t working"
      To result-field
End-If
```

Conditional Compilation

Similar to a (vastly) expanded "debugging lines" feature, this provides directives to enable specific lines of source to be included or omitted depending on the value of literals referenced in those directives.

Sample 10: Conditional Compilation

```
>>IF PROD-TEST = "TEST"
      Display "Some Var:" Some-Variable
      Perform Only-in-Test
*> the following evaluation is done at compile-time, not run-time
  >>If Too-Many-Errors > Error-Cnt
      Perform  Really-Bad
  >>Else
      >>DEFINE Error-Cnt AS Error-Cnt + 1
  >>END-IF
>>ELSE
  Perform Update-Production-Files
>>END-IF
```

In no way is this as powerful as the PL/I (or HLASM) macro language, but it does provide a way for a single source file to be used to compile different versions of similar programs.

Constants

Possibly the Enterprise COBOL compiler (when OPT is specified) is already doing all the optimization that "built-in" constants can do, but the new Standard should be able to get around the problem of OPT eliminating dummy eye-catchers in Working-Storage.

Sample 11: Constants

```
Working-Storage Section.
01  Start-of-WS   Constant Value
    "Working-Storage for Program: ABC Starts Here".
```

Note: This example also demonstrates that the PICTURE clause is optional – for an alphanumeric or national data-item with a VALUE clause (if you want the data-item to be as big – but no bigger – than the initial literal).

Note: The '02 Standard does NOT allow a "structure" (group item with subordinate elementary items) to be defined as a CONSTANT. However, this is being worked on for the '09 / next revision.

COPY and REPLACE "partial word" replacement

Although COBOL has supported a "trick" for getting partial word replacement (using parenthesis or colons in the COPY member) since the '85 Standard, the '02 Standard provides a "self-documenting" facility to do this – that also allows the original copy member to be used without REPLACING by some programs.

Sample 12: Partial word COPY REPLACING

```
COPYMEM *> source code of copy member is:

01  Group-Item
    05  GI-Field-1  Pic X.
    05  G1-Field-2  Binary-Char Signed.

*> COPY statement in main program is

Copy COPYMEM
    Replacing leading ==GI-== by ==WS-GI-==
              Trailing ==-2== by ==-Single-byte-binary==
      .
```

File Sharing and Record Locking (controlled in source code)

In today's IBM z/OS environment, one "deals" with sharing and locking via:

- JCL (DISP=)
- VSAM SHAREOPTIONS
- CICS
- Etc.

With the syntax introduced (as processor dependent – so a fully conforming implementation need not include support for it) an application program can internally control when and how to share and lock files and records – and what to do (retry, wait, skip) when a resource is already in use.

Sample 13: File Sharing / Record Locking

```
FILE-CONTROL. SELECT my-file ASSIGN TO accounts
...
LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORDS.

*> Additional options on I/O statements allow for selective actions
*> that are not normally needed, but may be useful in special circumstances:
READ ... ADVANCING ON LOCK:

*> Locked records are skipped
*> and the next unlocked record in sequential order is retrieved.
READ ... IGNORING LOCK:
```

HIGHEST-/LOWEST Algebraic Intrinsic Functions

With current (’85 Standard) COBOL it is difficult to initialize to or test if a numeric field (with decimal points or odd usages) is the highest possible value. For example

```
01 NumDec  Pic 99.99
Move all “9” to NumDec
```

Will NOT move “99.99” to NumDec, but rather will move “99.00” there. (The rules are obscure but are – and always have been – defined this way.) With the ’02 Standard it is now possible to get the “largest” or “smallest” possible value into a numeric field. The code looks like:

Sample 14: Use of HIGHEST-/LOWEST-Algebraic intrinsic functions

```
Repository. Function All Intrinsic.
      . . .
01 Num-Fields.
   05  N1  Pic S9(04)V99 Packed-Decimal. *> not even number of digits
   05  N2  Float-Short.
   05  N3  Pic $,$$9,999.99
   . . .
Move Highest-Algebraic to N1
If N2 = Lowest-Algebraic
Move lowest-algebraic to N3
```

Notice the omission of the word “function” because the correct REPOSITORY paragraph was used. Also, this shows how you can now use numeric intrinsic functions in a MOVE statement.

Although any of these statements could be accomplished with ’85 Standard code, it would be much more difficult and certainly harder to maintain when the data descriptions of the reference items were change.

INITIALIZE with new options

FILLER data items may be initialized with the INITIALIZE statement. Also, a VALUE phrase may be specified in the INITIALIZE statement to cause initialization of elementary data items to the literal specified in the VALUE clause of the associated data description entry (even if the data-item is defined in the Linkage or File sections – where no initialization is done at program initialization). For items of category data-pointer, object-reference, or program-pointer, the initialization is done to “NULL”.

It is important to note that none of this happens to existing source code; the new behavior requires new source code (for upward compatibility).

Sample 15: New INITIALIZE features

```
01 Group-Item.
   05                  Value “Implicit Filler”.
   05 Named-National   Value  NX”AB12CD98”.
   05 Filler      Pic 1 Value B”1”.
   05 Var-Tabl.
      10 Each-Elem Occurs 10 times
         Pic 9 Value 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (1) to (10).
. . .
 Initialize Group-Item
      With Filler
       All to Value
 .
```

Reversible order for conditional phrases

Hardly a “biggie” but the following source code will now be valid

Sample 16: Reversible order of conditional phrases

```
Compute ABC = XYZ + 1
    Not On size error
      Perform No-Truncation
    On Size Error
      Perform Too-big-to-handle
End-Compute
```

Select/Assign USING (for source code file specification)

All the recent releases of COBOL for the z/OS environment (but NOT for VM or VSE) have included the ability to “dynamically” specify either the DDName or the DSName (and attributes) for a file. This is done via the (C / POSIX) “putenv” routine. See:

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/igy3pg20/3.4.2.4

and

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/igy3lr20/4.2.3.1

(and following) for details on how to do this today.

However, with the ’02 Standard, programmers will be able to use a “native” COBOL method – that has been available (as an extension) in many COBOL products and in many environments – for years. For example (depending on how IBM were to actually implement this),

Sample 17: Dynamic File “name” Assignment

```
Select ABC  Assign Using Inp-Name.
Select XYZ  Assign Using Out-Name.
   . . .
01 Misc.
 05 Inp-Name
    10  Value “DDNAME=”
    10  Dynamic-Name    Pic X(8)
 05 Out-Name Value
    “DSN=HLQ.NEWQSAM,DISP=(NEW,CATLG,DELETE),”-
    “UNIT=SYSDA,DCB=(RECFM=F,LRECL=80),”-
    “SPACE=(CYL(2,1))”
 . . .
If Week-Day
   Move “WEEKDDN” to Dynamic-Name  *> must have WEEKDN in JCL as DDNAME
Else
   Move “NOTWORK” to Dynamic-Name
End-IF
Open Input ABC
     Output XYZ
```

VALUE clauses for individual Table entries

(As demonstrated in the INITIALIZE example above)

It is now possible to provide specific values for individual items – or groups of items in a table definition – without any REDEFINES or PERFORM loop. (Of course we all know <G>, that with the ’85 Standard and Enterprise COBOL it is already possible to initialize every occurrence of a table entry to the same value, e.g. ZEROES for each individual numeric item within a table – simply by a single VALUE clause under the OCCURS statement.)

The ’02 Standard allows one to provide (for example) the first 5 occurrences of an item to be 1, 2, 3, 4, and 5 and all remaining items to be zero.

This feature is a little complex to explain, but I suspect, it will be widely used – when available. The following shows some of its capabilities.

Sample 18: VALUE clauses for individual Table entries

```
05    Level-1  Occurs 10 times.
   10  Level-2 Occurs 3 times PIC X
                  Value “A” “B” “C” From (2 2)  to (3 1)
                        ”X” “Y” “Z” From (1 1) to (1 3)
                         Space      from (3 2).
```

Web-Pages of (possible) Interest

For additional information on any of the documents or sites listed below or to get on the "notification" list for public reviews of future Technical Reports (TR's) and the next ISO COBOL Standard (currently expected in 2008), send an email to the J4 chair at:

Don.Schricker@microfocus.com

- To obtain a downloadable PDF copy of the ANSI version of ISO 1989:2002 Standard (for a minimal price), see:

  http://www.techstreet.com/cgi-bin/basket?action=add&item_id=3574917

- To obtain a printed copy of the ANSI version of the ISO 1989:2002 Standard (for a still minimal price), see:

  http://www.techstreet.com/cgi-bin/basket?action=add&item_id=3574918

- To obtain a downloadable or CD-ROM version of the ISO version of ISO 1989:2002 Standard (for a *not* so reasonable price), see:

  http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=28805

- For a copy of the (Dec 02, 2004) "Technical Corrigendum 1" to ISO 1989:2002, see:

  http://www.cobolportal.com/j4/files/04-0216.doc

- For a copy of the Draft (Feb 4, 2005) "Technical Corrigendum 2" to ISO 1989:2002, see:

  http://www.cobolportal.com/j4/files/05-0016.doc

- For a copy of the (final draft) of the (now approved) Technical Report on "Object finalization for programming language COBOL", see:

  http://www.cobolportal.com/j4/files/03-0046.doc

- For a current (Aug 18, 2005) draft of the "Native COBOL Syntax for XML Support", see:

  http://www.cobolportal.com/j4/files/05-0172.doc

- For a current (as of Feb 11, 2005) draft of the "Collection Class Library for programming language COBOL", see:

  http://www.cobolportal.com/j4/files/05-0042.doc

- For "COBOL Standardization Process – Status," see:

  http://www.cobolstandards.com/

- For access to ALL of the current (and recent past) documents being (or having been) processed by J4, see:

  http://www.cobolportal.com/j4/index.asp?bhcp=1

- For (international) WG4 status, see:

  http://www.cobolstandard.info/wg4/wg4.html

In Conclusion

Even if every current customer of IBM's were to ask IBM to implement every feature, change, and enhancement in the ISO 2002 COBOL Standard, I don't know if/when IBM would do so. However, as this presentation has demonstrated, IBM has already met many of its customers' identified needs by implementing some features of the new Standard while delaying implementation of other features that may or may not actually be desired or desirable.

If there are specific features that are not currently implemented but that seem desirable to your installation (or as if they would meet existing or foreseen business needs), it is important that you communicate this to IBM – via SHARE requirements and your local IBM marketing branch. Similarly, although it is unlikely that IBM needs a "do *NOT* implement" requirement for undesirable features, it may be worth your while to communicate your views on such changes sooner than later – to avoid eventually paying for the resources need to develop features that you do not want or need.

Appendix A: Substantive Changes - changes not affecting existing programs

This section represents the analysis of Bill Klein (wmklein@ix.netcom.com) only. It should *NOT* be read as reflecting (necessarily) the views, analysis, or opinions of IBM, SHARE, or any other group or individual.

Reader Beware !!!

Appendix A.1: Items currently Available/Implemented in IBM Enterprise COBOL V3

Items listed below with "(RW)" are part of the (*required* in the ISO 2002 Standard) "Report Writer facility" of the ISO 2002 Standard. These features are currently available to any IBM site licensed for (and using) both IBM Enterprise COBOL and the "Report Writer" add-on product.

The numbers at the beginning of each item reflect the numbering within the ISO 2002 COBOL Standard, in the section:

F.2 Substantive changes not affecting existing programs

2 ACCEPT statement

The capability to access the four-digit year of the Gregorian calendar is added to the ACCEPT statement.

3 Apostrophe as quotation symbol

The apostrophe character as well as the quotation mark may be used in the opening and closing delimiters of alphanumeric, boolean, and national literals. A given literal may use either the apostrophe or the quotation mark, but both the starting and ending characters are required to be the same. Whichever character is used, it is necessary to double that character to represent one occurrence of the character within the literal. Both formats may be used in a single source element.

5 Arithmetic operators

No space is required between a left parenthesis and unary operator or between a unary operator and a left parenthesis.

6 AT END phrase

The AT END phrase of the READ statement does not have to be specified if there is no applicable USE statement.

10 CALL argument level numbers

CALL arguments may be elementary or groups with any level number. Formerly, they had to be elementary or have a level number of 1 or 77.

12 CALL parameter defined with OCCURS DEPENDING ON

For an argument or formal parameter defined with OCCURS DEPENDING ON, the maximum length is used.

13 CALL parameter length difference

The size of an argument in the USING phrase of the CALL statement may be greater than the size of the matching formal parameter if either the argument or the formal parameter is a group item. Formerly, the sizes were required to be the same.

14 CALL recursively

The capability of calling an active COBOL program has been added to COBOL.

20 CODE clause (RW)

The identifier phrase is provided in the CODE clause in the report description entry.

25 COLUMN clause (RW)

A relative form is provided using PLUS integer, by analogy with LINE; COLUMN RIGHT and COLUMN CENTER are provided, allowing alignment of a printable item at the right or center; and COL, COLS, and COLUMNS are allowed as synonyms for COLUMN.

26 COLUMN, LINE, SOURCE and VALUE clauses (RW)

These clauses may have more than one operand in a report group description entry.

27 Comment lines anywhere in a compilation group

Comment lines may be written as any line in a compilation group, including before the identification division header.

35 Control data-name (RW)

This is allowed to be omitted on TYPE CH/CF if only one control is defined.

36 Conversion from 2-digit year to 4-digit year

There are three functions for converting from a 2-digit year to a 4-digit year. DATE-TO-YYYYMMDD, DAY-TO-YYYYDDD, and YEAR-TO-YYYY convert from YYnnnn to YYYYnnnn, YYnnn to YYYYnnn, and YY to YYYY, respectively.

37 COPY statement

An alphanumeric literal may be specified in place of text-name-1 or library-name-1.

38 COPY statement

When more than one COBOL library is referenced, text-name need not be qualified when the library text resides in the default library.

39 COPY statement

The ability to nest COPY statements is provided. Library text incorporated as a result of processing a COPY statement without a REPLACING phrase may contain a COPY statement without a REPLACING phrase.

44 Currency sign extensions

The CURRENCY SIGN clause has been extended to allow for national characters and for multiple distinct currency signs, which may have any length.

46 DISPLAY statement

If the literal in a DISPLAY statement is numeric, it may be signed.

50 EXIT PROGRAM allowed as other than last statement

EXIT PROGRAM is allowed to appear as other than the last statement in a consecutive sequence of imperative statements.

53 FILLER (RW)

FILLER is allowed in the report section.

55 Fixed-point numeric items

The maximum number of digits that may be specified in a numeric literal or in a picture character-string that describes a numeric or numeric-edited data item is increased from 18 to 31.

61 Global clause in the linkage section

The GLOBAL clause may be specified in level 1 data description entries in the linkage section.

62 GOBACK statement

A GOBACK statement has been added that always returns control, either to the operating system or to the calling runtime element.

63 Hexadecimal Literals

The ability was added to specify alphanumeric, boolean, and national literals using hexadecimal (radix 16) notation.

70 Index data item

The definition of an index data item may include the SYNCHRONIZED clause.

77 Intrinsic function facility

Previously, the intrinsic function facility was a separate module and its implementation was optional. The intrinsic function facility is integrated into the specification and it shall be implemented by a conforming implementation.

78c Intrinsic functions

New intrinsic functions are:

DATE-TO-YYYYMMDD

DAY-TO-YYYYDDD

YEAR-TO-YYYY

79 INVALID KEY phrase

The INVALID KEY phrase does not have to be specified if there is no applicable USE statement.

80 Local-Storage Section

A facility was added to define data that is set to its initial values each time a function, method, or program is activated. Each instance of this source element has its own copy of this data.

83 OCCURS clause (RW)

Repetition vertically or horizontally and a STEP phrase are added for report writer.

85 Optional word OF (RW)

Allowed after SUM.

86 Optional words FOR and ON (RW)

Allowed after TYPE CH or CF.

87 OR PAGE phrase of the CONTROL HEADING phrase (RW)

This enables the control heading group to be printed at the top of each page as well as after a control break

88 PAGE FOOTING report group (RW)

Such a group is allowed to have all relative LINE clauses.

89 PAGE LIMIT clause (RW)

New COLUMNS phrase is provided to define maximum number of horizontal print positions in each report line and a LAST CONTROL HEADING phrase was added.

90 Paragraph-name

A paragraph-name is not required at the beginning of the procedure division or a section.

92 PERFORM statement

A common exit for multiple active PERFORM statements is allowed.

94 PICTURE clause

The maximum number of characters that may be specified in a picture character-string is increased from 30 to 50.

96b PICTURE clause (RW)

The PICTURE clause may be omitted when the VALUE clause in a ... report group description entry specified with an alphanumeric, boolean, or national literal.

100 PICTURE clause

The symbol 'N' can be used in a PICTURE character-string to specify a national or a national-edited data item.

102 PLUS and MINUS (RW)

The symbol + or - is synonymous with PLUS or MINUS, respectively, in the COLUMN and LINE clauses.

103 Pointer data

A new class of data is introduced, a pointer type which holds data and program addresses in a machine-specific or system-specific way.

104 PRESENT WHEN clause (RW)

The PRESENT WHEN clauses allows lines and printable items, or groups of them, to be printed or not, depending on the truth value of a condition.

105 Program-names as literals

The option to specify a literal as the program-name to be externalized was added for names that are not valid COBOL words or need to be case-sensitive.

115 Report writer (RW)

Previously, the report writer was a separate module and its implementation was optional. The report writer facility is integrated into the specification and it shall be implemented by a conforming implementation. In addition, the following changes were made and are documented elsewhere in the list of substantive changes not affecting existing programs:

- CODE IS identifier added
- COLUMN clause has several additions
- COLUMN, LINE, SOURCE, and VALUE clauses are allowed to have more than one operand
- Control data-name can be omitted
- FILLER allowed in report section
- National characters are allowed in reports
- OCCURS allows repetition vertically or horizontally and a STEP phrase
- Optional words OF, FOR, and ON in some clauses
- OR PAGE phrase of CONTROL HEADING phrase
- PAGE FOOTING allowed to have only relative LINE clauses
- PAGE LIMIT includes COLUMN phrase
- PLUS or MINUS can be the symbol '+' or '-', respectively
- PRESENT WHEN clause added
- SIGN clause does not require SEPARATE phrase in report section
- SOURCE allows arithmetic expressions and functions and a ROUNDED phrase

- SUM clause has many changes
- USE BEFORE REPORTING changed
- VARYING clause may be used with an OCCURS clause or a multiple LINE or multiple COLUMN clause
- WITH RESET phrase added to NEXT PAGE

116 Report writer national character support (RW)

The capability of printing national characters and alphanumeric characters in a report is provided.

123 SIGN clause in a report description entry (RW)

The SEPARATE phrase is no longer required in a report description entry and the SIGN clause may be specified at the group level.

126 SORT statement

GIVING files in a SORT statement may now be specified in the same SAME RECORD AREA clause.

127 SOURCE clause in a report description entry (RW)

The sending operand may be a function-identifier.

128 SOURCE clause in a report description entry (RW)

An arithmetic-expression is allowed as an operand and a ROUNDED phrase was added.

134 SUM clause in a report description entry (RW)

The SUM clause was extended in the following ways:

- Extension to total a repeating entry.
- Now allowed in any TYPE of report group, not only control footing.
- SUM of arithmetic-expression format.
- Checks for overflow of a sum counter during totaling.
- Any numeric report section item may be totaled, not just another sum counter.
- ROUNDED phrase.

140 USE BEFORE REPORTING

The effect of GLOBAL in a report description and a USE declarative is further elucidated.

145 VALUE clause ignored in external data items and in linkage and file sections

The data-item and table formats of the VALUE clause may be specified in data descriptions in the linkage section and in the file section and for items described with the EXTERNAL clause. These VALUE clauses are ignored except during the execution of an explicit or implicit INITIALIZE statement.

146a VARYING clause (RW)

A VARYING clause is provided in the ... report writer facilities to be used in conjunction with an OCCURS clause.

147 WITH RESET phrase (RW)

This was added to the NEXT PAGE phrase of the NEXT GROUP clause, to reset PAGE-COUNTER back to 1.

Appendix A.2: Items with comparable functionality in IBM Enterprise COBOL V3R3

7b Floating point data (Processor Dependent)

See SHARE Requirement: **SSLNGC041361**

Two new representations of numeric data type are introduced, ... The floating-point type exists both in a numeric form, with a machine-specific representation, and in a numeric-edited form.

29a Compiler directives

See SHARE Requirement: **SSLNGC031359**

Compiler directives instruct the compiler to take specific actions during compilation.

Compiler directives are provided:

- to control the source listing
- to request flagging of syntax that might be incompatible between the previous COBOL standard and the current standard
- to specify page ejection

40 COPY statement

A SUPPRESS PRINTING phrase is added to the COPY statement to suppress listing of library text incorporated as a result of COPY statement processing.

47 Dynamic storage allocation

ALLOCATE and FREE statements are provided for obtaining storage dynamically. This storage is addressed by data-pointers.

99 PICTURE clause.

See SHARE Requirement: **SSLNGC041361**

The symbol 'E' can be used in a PICTURE character-string to specify a floating-point numeric edited data item.

119 SELECT clause.

A file may be dynamically assigned by specifying a data-name in the SELECT clause.

Appendix A.3: Items Partially Implemented in IBM Enterprise COBOL V3R3

19 COBOL words reserved in context

Certain words added to the COBOL standard are reserved only in the contexts in which they are specified and were not added to the reserved word list.

67 Implicit qualification of data-names within the same group.

067

Data-names referenced in a data description entry need not be qualified when they are defined in the same group as the subject of the entry, unless qualifiers are needed to establish uniqueness within that group.

78a Intrinsic functions

See SHARE Requirement **SSLNGC041361**

New intrinsic functions are:

- DISPLAY-OF
- NATIONAL-OF

81 National character handling

The capability is added for using large character sets, such as ISO/IEC 10646-1, in source text and library text and in data at execution time. Class national and categories national and national-edited are specified by picture character-strings containing the symbol 'N'; national literals are identified by a separator N'', N', NX'', or NX'. Usage national specifies representation of data in a national character set. User-defined words may contain extended letters. Processing of data of class national is comparable to processing data of class alphanumeric, though there are some minor differences. Conversions between data of classes alphanumeric and national are provided by intrinsic functions.

82 Object orientation

Support for object oriented programming has been added.

Appendix A4: Items not yet implemented by IBM

Items listed below with "(PD)" are in the "Processor Dependent" list of the ISO 2002 Standard. Therefore, IBM need *not* implement them, to provide a "fully conforming" compiler.

1 ACCEPT screen (PD)

The ACCEPT statement is extended to allow a screen item specified in the screen section to be used for operator input at a terminal.

4 ARITHMETIC clause

See SHARE Requirement: **SSLNGC041361**

The STANDARD phrase specifies that certain arithmetic will be performed in a well-defined manner and may yield results that are portable. When standard arithmetic is in effect there is no restriction on composite of operands.

7a BINARY data (PD)

See SHARE Requirement: **SSLNGC031358**

Two new representations of numeric data type are introduced, a binary representation which holds data in a machine-specific way and is not restricted to decimal ranges of values, …

8 Bit/boolean support

See SHARE Requirement: **SSLNGC041361**

The capability of defining bit strings and setting or testing boolean values is added. Data of class and category boolean may be represented as bits, alphanumeric characters, or national characters by specifying usage bit, display, or national, respectively. Boolean data items are specified by picture character strings containing the symbol '1'; boolean literals are identified by a separator B'', B', BX'', or BX'. Boolean operations BOOR, B-AND, B-OR, and B-NOT are provided for use in boolean expressions.

9 Boolean functions

See SHARE Requirement: **SSLNGC041361**

The following intrinsic functions are provided for processing boolean items: BOOLEAN-OF-INTEGER and INTEGER-OF-BOOLEAN. These functions convert between numeric and boolean items.

11 CALL BY CONTENT parameter difference

See SHARE Requirement: **SSLNGC031358**

A parameter passed by content does not have to have the same description as the matching parameter in the called program.

15 CALL statement

Arithmetic expressions and literals may be used as arguments in a new format of the CALL statement.

16 Class condition test with alphabet-name

The content of a data item can be tested against the coded character set named as alphabet-name.

17 Class condition test with NUMERIC

The rule that prohibited a group data item from containing a numeric item with an operational sign was removed.

18 COBOL word

The maximum length of a COBOL word is increased from 30 to 31 characters.

21 Coded character sets for national character data

Predefined alphabet-names UCS-4, UTF-8, and UTF-16 are added to the SPECIAL-NAMES paragraph for referencing the Universal Multiple-Octet Coded Character Set (UCS). Additional alphabet-names for national coded character sets can be defined by the implementor or by the programmer in the SPECIAL-NAMES paragraph.

22 CODE-SET clause (PD)

A code set defined with literals in the SPECIAL-NAMES paragraph can be specified in the CODE-SET clause for use in conversion on input and output of data.

23 CODE-SET clause (PD)

A CODE-SET clause may specify an alphabet defining a single-octet (alphanumeric) coded character set, such as ISO/IEC 646, and multiple-octet (national) coded character set, such as ISO/IEC 10646-1.

24 Collating sequences for national character data

A predefined alphabet, UCS-4, is added to the SPECIAL-NAMES paragraph for use in referencing a collating sequence associated with ISO/IEC 10646-1 and ISO/IEC 10646-2. Additional alphabet-names for national collating sequences can be defined by the implementor or by the programmer in the SPECIAL-NAMES paragraph. For each use of a collating sequence, language is provided for specifying either an alphanumeric collating sequence or a national collating sequence, or both.

28 Common exception processing

See SHARE Requirement: **SSLNGC041361**

The user may select exception checking for violations of general rules in the standard, such as subscripts out of bounds, reference modifiers out of bounds, CALL parameter mismatches, etc. This is enabled by the TURN compiler directive and changes to the USE statement.

29b Compiler directives

See SHARE Requirement: SSLNGC031359

Compiler directives instruct the compiler to take specific actions during compilation.

Compiler directives are provided:

- to cause propagation of an exception condition to an activating runtime element
- to specify whether the reference format of the source text or library text that follows is fixed-form or free-form
- to turn checking for certain exception conditions on or off:
- to conditionally treat certain text lines as comments
- to define values that may be used in constant entries in the data division
- to allow for the reporting of leap seconds for specifying options that are defined by the implementor
- to request the flagging of syntax that might require native arithmetic when standard arithmetic is in effect.

30 Computer-name in SOURCE-COMPUTER and OBJECT COMPUTER paragraphs

The computer-name may be omitted even when additional clauses are specified.

31 Concatenation expression

A concatenation expression operates on two literals of the same class to concatenate their values.

31 Conditional compilation

Directives are provided to enable specific lines of source to be included or omitted depending on the value of literals referenced in those directives.

33 Conditional phrases

The conditional phrases in the ACCEPT, ADD, CALL, COMPUTE, DELETE, DISPLAY, DIVIDE, MULTIPLY, READ, RECEIVE, REWRITE, START, STRING, SUBTRACT, UNSTRING, and WRITE statements may be specified in any order.

34 Constants

The user may define constants in the data division with constant entries.

41 COPY and REPLACE statement partial word replacement

See SHARE Requirement: **SSLNGC031359**

LEADING and TRAILING phrases of the REPLACE statement and the REPLACING phrase of the COPY statement allow replacement of partial words in source text and library text. This is useful for prefixing and postfixing names.

42 Cultural adaptability and multilingual support (PD)

Support is provided for using local conventions that depend on language and culture (cultural conventions). A "locale" contains the specification of cultural conventions. Features supported are collating sequences, date and time formats, monetary and number formats, and character case mappings.

43 Cultural convention switching

The capability of switching the set of cultural conventions, known as locales, in effect at runtime is provided by the SET statement.

45 DISPLAY screen (PD)

The DISPLAY statement is extended to allow a screen item specified in the screen section to be used for output to the operator of a terminal.

48 EVALUATE statement, partial expressions

A partial expression may now be used as a selection object in an EVALUATE statement. This partial expression, when combined with its corresponding selection subject, forms a complete conditional expression.

49 EXIT statement

See SHARE Requirement: **SSLNGC031358**

The ability to immediately exit an inline PERFORM statement, a paragraph, or a section has been added.

51 EXTERNAL AS literal

The option to specify a literal in the EXTERNAL clause was added for external names that are not valid COBOL words or need to be case sensitive.

52 File sharing (PD)

File sharing provides a cooperative environment that allows the user to permit or deny access by concurrent file connectors to a physical file.

54 Fixed-form reference format area A and B removed

Areas A and B of the previous COBOL standard reference format have been combined into one area called the program-text area. Restrictions of area A and area B have been removed. The previous COBOL standard reference format is fully upward compatible.

56 FLAG-85 directive

A directive, FLAG-85, has been added that causes compiler flagging of language elements that may be incompatible between the previous COBOL standard and this International Standard.

57 FORMAT clause

For sequentially-organized files, the FORMAT clause provides the capability to present data so that people may read it. This is called external media format; it includes any special encoding needed by the operating environment to successfully print or display data, as might be needed for multiple-octet data, for example.

58 Free-form reference format

Free-form reference format provides varying-length source lines and permits source text and library text to be written with a minimum of restrictions on the placement of source code on a line. A compiler directive permits selection and intermixing of free-form or fixed-form reference format, with fixed-form as the default.

59 FUNCTION keyword

The keyword FUNCTION may be omitted from a function reference when the referenced function-name or the ALL phrase is specified in the REPOSITORY paragraph.

60 Function use expanded

See SHARE Requirement: **SSLNGC031359**

The restrictions that numeric and integer functions could be used only in arithmetic expressions was removed as well as the restriction that no integer function could be used where an unsigned integer was required. Numeric and integer functions may now be used anywhere a numeric sender is allowed and an integer form of the ABS function may now be used where an unsigned integer is required.

64 HIGHEST-ALGEBRAIC and LOWEST-ALGEBRAIC functions

The HIGHEST-ALGEBRAIC and LOWEST-ALGEBRAIC functions provide the ability to manipulate numeric data items in a manner similar to the means that HIGH-VALUES and LOW-VALUES permit with alphanumeric data items, but without the risks of the data incompatibilities associated with those figurative constants.

65 Identification division header

The identification division header is now optional.

66 Implementor-defined successful I-O status codes

See SHARE Requirement: **SSLNGC041361**

A range of I-O status codes is allocated for use by the implementor to indicate conditions associated with successful completion of an input-output statement.

68 Inline comment

A comment may be written on a line following any character-strings of the source text or library text that are written on that line. An inline comment is introduced by the two contiguous characters '*>'.

69 Index data item

An index data item may be referenced as an argument in a BYTE-LENGTH, LENGTH, MAX, MIN, ORD-MAX, or ORD-MIN function.

71 Initialization of tables

The VALUE clause may be used to initialize selected elements of a table to specific values.

72 INITIALIZE statement, FILLER phrase

See SHARE Requirement: **SSLNGC031359**

FILLER data items may be initialized with the INITIALIZE statement.

73 INITIALIZE statement, VALUE phrase

See SHARE Requirement: **SSLNGC031359**

A VALUE phrase may be specified in the INITIALIZE statement to cause initialization of elementary data items to the literal specified in the VALUE clause of the associated data description entry or to NULL for items of category data-pointer, object-reference, or program-pointer.

74 INITIALIZE statement, variable-length tables

See SHARE Requirement: **SSLNGC031359**

A variable-length table may be initialized with the INITIALIZE statement.

75 INSPECT CONVERTING statement

A figurative constant beginning with ALL may be specified as the TO literal in the INSPECT CONVERTING statement.

76 INSPECT CONVERTING statement

The same character in the data item referenced by the identifier to the left of TO or the literal to the left of TO may appear more than once. If a character is duplicated, the first occurrence is used in the substitution and any repetitions are ignored.

78b Intrinsic functions

See SHARE Requirement: **SSLNGC031359**

New intrinsic functions are:

- TEST-DATE-YYYYMMDD
- TEST-DAY-YYYYDDD
- TEST-NUMVAL
- TEST-NUMVAL-C
- TEST-NUMVAL-F

78d Intrinsic functions

New intrinsic functions are:

- ABS
- BOOLEAN-OF-INTEGER
- BYTE-LENGTH
- CHAR-NATIONAL
- E
- EXCEPTION-FILE
- EXCEPTION-FILE-N
- EXCEPTION-LOCATION
- EXCEPTION-LOCATION-N
- EXCEPTION-STATEMENT
- EXCEPTION-STATUS
- EXP
- EXP10
- FRACTION-PART
- HIGHEST-ALGEBRAIC
- INTEGER-OF-BOOLEAN
- LOCALE-COMPARE
- LOCALE-DATE
- LOCALE-TIME
- LOWEST-ALGEBRAIC
- NUMVAL-F
- PI
- SIGN
- STANDARD-COMPARE

84 OCCURS clause, KEY phrase

An entry between a data description entry that contains an OCCURS clause and the entry describing a data item referenced in the KEY phrase of the OCCURS may itself contain an OCCURS clause, as long as it is not a group item in the hierarchy of the data item referenced in the KEY phrase.

91 PERFORM statement.

See SHARE Requirement: **SSLNGC031358**

The AFTER phrase is allowed in an inline PERFORM.

93 PERFORM VARYING statement

The BY value may be 0, the FROM value is not required to correspond to a table element at the start of the execution of the PERFORM statement, and augmentation on an index can produce a value outside of the range of the associated table.

95 PICTURE clause

The content of a character position described with the picture symbol ‘A’ is not required to be a letter except in the format stage of a VALIDATE statement.

96a PICTURE clause

The PICTURE clause may be omitted when the VALUE clause in a data description ... description entry is specified with an alphanumeric, boolean or national literal.

97 PICTURE clause

The currency symbol may be specified at the end of a PICTURE character-string, optionally followed by one of the symbols '+', '-', 'CR', or 'DB'.

98 PICTURE clause

See SHARE Requirement: **SSLNGC041361**

The symbol '1' can be used in a PICTURE character-string to specify a boolean data item.

106 Program prototypes

See SHARE Requirement: **SSLNGC031358**

The interface and characteristics of a program are defined in a program-prototype. This permits passing parameters by value, omission of parameters, type checking and coercion of arguments, and the use of calling conventions other than those used by default for COBOL.

107 Qualification limits

The previous limit of fifty qualifiers has been removed.

108 Qualification of index-names

Index-names may be qualified in the same manner as data and condition-names, even in cases where uniqueness of reference may already be established.

109 READ PREVIOUS (PD)

See SHARE Requirement: **SSLNGC031359**

The READ statement has been enhanced to allow reading the record prior to that pointed to by the file position indicator.

110 RECORD KEY and ALTERNATE RECORD KEY (PD)

Keys for indexed files may be made up from more than one component.

111 Record locking (PD)

Record locking allows control of record access for shared files. A record lock is used to prevent access to a record from other file connectors.

112 RELATIVE KEY clause

A RELATIVE KEY clause may be specified by itself in a file control entry. Previously, RELATIVE KEY was required to be specified as a phrase of the ACCESS MODE clause.

113 REPLACE statement syntax relaxation

A REPLACE statement may be specified anywhere in a compilation unit that a character-string or a separator, other than the closing delimiter of a literal, may appear. Previously, a REPLACE statement was required to follow a separator period when it was other than the first statement in an outermost program.

114 REPLACE statement nesting

A REPLACE statement may be specified and terminated without canceling the effect of a previous REPLACE statement.

117 SAME AS clause

The SAME AS clause indicates that the description of a data item is identical to that of another item.

118 Screen section (PD)

The screen section provides a non-procedural means of declaring screen items that are to appear on a terminal, their position, and various attributes. The screen items may be used for input or output and be associated with data items described in other sections within the data division.

120 SELECT WHEN clause (PD)

The SELECT WHEN clause of a record description permits selection of one of several record description entries in the file section during input-output operations. The selected record description entry is used with a CODE-SET clause or a FORMAT clause to process individual data items in a record.

121 SET index-name

The result of setting an index can be outside of the range of the associated table. Also, the setting of an index in the TO phrase does not have to be within the range of the associated table.

122 SET screen-name (PD)

The SET statement is extended to allow screen item attributes to be dynamically specified.

124 SIGN clause on group items

A sign clause may be specified on any group item. Formerly, the group had to contain at least one numeric item.

125 SORT statement

See SHARE Requirement: **SSLNGC031359**

A SORT statement may be used to sort a table. This sort may be done using the fields specified in the KEY phrase defining the table, by using the entire table element as the key, or by using specified key data items.

129 START FIRST, LAST, and LESS THAN (PD)

See SHARE Requirement: **SSLNGC031359**

The START statement has been enhanced to allow starting to a position before the key and to point to the first or last record in the file.

130 START and sequential files

See SHARE Requirement: **SSLNGC031359**

The START statement has been enhanced to allow a sequential file if the FIRST or LAST phrase is specified.

131 STOP WITH STATUS (PD)

See SHARE Requirement: **SSLNGC031359**

The WITH STATUS phrase was added to the STOP statement to allow a run unit to return a value to the operating system.

132 STRING statement

The DELIMITED phrase is optional in the STRING statement. DELIMITED BY SIZE is assumed.

133 Subscripting with arithmetic expressions

An arithmetic expression may now be used as a subscript, not just the forms data-name + integer and data-name – integer.

135 System-names

Computer-names no longer have to be different from other types of system-names. Text-names and library-names are now system-names, instead of user-defined words, and do not have to be different from user-defined words or other types of system-names.

136 THROUGH phrase in VALUE clause and EVALUATE statement

A collating sequence can be identified in the THROUGH phrase for use in determining a range of values. This allows use of portable ranges across various implementations. It also allows the range to be determined from a locale, which is not necessarily portable when different locales are used.

137 TYPE, and TYPEDEF clauses

The TYPEDEF clause identifies a type declaration which creates a user-defined type. The TYPE clause is used to apply this user-defined type to the description of a data item. No storage is allocated for the type declaration.

138 Underscore (_) character.

See SHARE Requirement: **SSLNGC03002**

The basic special characters of the COBOL character repertoire have been expanded to include the underscore (_) character, which can be used in the formation of COBOL words.

139 UNSTRING statement

The sending operand may be reference modified.

141 USE statement syntax

The word PROCEDURE, previously required, is optional.

142 User-defined functions

The ability was added to write functions that are activated in a manner similar to intrinsic functions. The word FUNCTION is not specified as part of this invocation.

143 Validate facility

See SHARE Requirement: **SSLNGC041360**

The new statement VALIDATE was added, giving the ability to perform comprehensive data validation. The new data division clauses CLASS, DEFAULT, DESTINATION, INVALID, PRESENT WHEN, VALIDATE-STATUS, and VARYING were also added, with the PRESENT WHEN and VARYING clauses having a similar function to those in the report section. These clauses, together with the VALID, INVALID and WHEN phrases of the level-88 VALUE clause, are ignored when they appear in a data description that is the operand of any statement other than VALIDATE.

144 VALUE clause, WHEN SET TO FALSE phrase in data division

See SHARE Requirement: **SSLNGC031359**

The WHEN SET TO FALSE phrase allows specification of a FALSE condition value. This value is moved to the associated conditional variable when the SET TO FALSE statement is executed for the associated condition-name.

146b VARYING clause

See SHARE Requirement: **SSLNGC041360**

A VARYING clause is provided in the validate ... facilities to be used in conjunction with an OCCURS clause.

148 Writing literals to a file

A record can be written from a literal using the WRITE, RELEASE, or REWRITE statement. Using the FILE phrase of WRITE, alphanumeric, national, and boolean literals and the figurative constant SPACE can be written. When the FILE phrase is not used, any literal can be written.

149 Writing without a record-name

A FILE phrase on the WRITE and REWRITE statements permit writing records from working storage without having a corresponding record description entry in the file section.

Appendix B: FLAG-85 directive and other changes "Potentially Affecting"

The FLAG-85 directive specifies options to flag certain syntax that might be incompatible between the previous COBOL standard and the current COBOL standard.

This is similar to the (previously available – but dropped in Enterprise COBOL) FLAGMIG compiler option. However, this is intended to detect (and flag) any syntax that may compile in ANY '85 Standard compiler but will yield potentially (but not necessarily) different results at run-time.

I have not done a detailed analysis of these item (nor – to the best of my knowledge has IBM) to determine which – if any will actually produce different results when/if IBM produces an ISO 2002 conforming compiler. My initial (and possibly mistaken) belief is that IBM currently (in Enterprise COBOL) works as required by the 2002 Standard – not as was "required" or "ambiguous" in the 1985 Standard.

In addition to these specific items, there are a total of 40 items that would need to be "studied in depth" to determine how much (if any) of a migration inhibitor they would be from the section of the ISO 2002 Standard called,

Substantive changes potentially affecting existing programs

One topic of particular interest (to some IBM customers) is the fact that IBM has previously indicated that they have no CURRENT intention to "remove support" for any of the items identified as "OBSOLETE" in the 1985 Standard and which have been removed from the 2002 Standard. Instead, they have indicated that when/if they provide a 2002 conforming compiler they would continue to provide support for such features as "IBM extensions."

On the other hand, such obsolete features as "ALTER" and "DEBUG-ITEM" have limited support in IBM's current OO and recursive environments, so programmers should NOT assume that the obsolete features will be "enhanced" to integrate fully with new functionality.

CORRESPONDING

In an ADD, MOVE, or SUBTRACT statement with the CORRESPONDING phrase, if subscripting with other than a constant is specified on any of the operands, the statement shall be flagged.

DE-EDITING

A de-editing MOVE statement shall be flagged.

DIVIDE

In a DIVIDE statement with the REMAINDER phrase, if the quotient data item is not described with a sign (there is no S in the PICTURE clause) and either the divisor or dividend is described with a sign, the DIVIDE statement shall be flagged.

FUNCTION-ARGUMENT

If the intrinsic function RANDOM is followed immediately by an arithmetic expression that is enclosed in parentheses, and the function is specified as an argument in an intrinsic function that allows multiple arithmetic expressions as arguments, the function RANDOM shall be flagged.

MOVE

If an alphanumeric literal or data item is moved to a numeric data item and the number of characters in the sending operand is greater than 31, the MOVE statement shall be flagged.

NUMVAL

The NUMVAL and NUMVAL-C intrinsic functions shall be flagged.

SET

If a condition-setting format SET statement references a variable length group item, the SET statement shall be flagged.

STANDARD-1

If STANDARD-1 is specified in an ALPHABET clause in the SPECIAL-NAMES paragraph, the ALPHABET clause shall be flagged.

STANDARD-2

If STANDARD-2 is specified in an ALPHABET clause in the SPECIAL-NAMES paragraph, the ALPHABET clause shall be flagged.

ZERO-LENGTH

A READ statement that could return a zero-length item or any statement that references a data item that could be a zero-length item shall be flagged.

Appendix C: Introduction to the Validate facility

The information at this site has been excerpted from the “Concepts” Annex of the ISO 2002 COBOL Standard. The section on ***VALIDATE*** begins on page 792.

E.19 Validate facility

The validate facility provides a major procedural statement VALIDATE and several associated data division clauses that enable data to be checked for various errors and inconsistencies in a high-level and comprehensive manner.

The VALIDATE statement can be used to perform validation on any data item defined in the file, working-storage, local-storage, or linkage section. The item is checked for conformance to its data description and messages or flags of the programmer's choice may be issued or set in response. Data items can also be stored automatically in target locations.

The detailed processing undertaken by the VALIDATE statement is established entirely by the description of the referenced data item rather than by code in the procedure division. As well as general data division clauses, such as PICTURE, additional clauses may be included that control the action of the VALIDATE statement, but have no effect otherwise.

The data item may be a group or elementary item of any length and complexity. The operation of the VALIDATE statement is divided into several stages. Each stage is completed for the entire data item, including all its subordinate items, before the next stage begins. If an elementary data item fails one stage of validation it cannot be rejected at any further stage. If a check fails, there is no interruption to processing: instead, errors are indicated by the storing of messages or indicators defined by the programmer. The stages are as follows:

— Format validation

— Input distribution

— Content validation

— Relation validation

— Error indication

E.19.1 Format validation

This stage uses the PICTURE clauses, possibly modified in their meaning by the SIGN and USAGE clauses, to check that each data item has the expected data format. An elementary item is assigned a default value if it fails this check or if its usage is display or national and its value is all spaces. The default value is used in subsequent references to the item. A group item may also have a DEFAULT clause.

The clauses relevant to this stage are: PICTURE, DEFAULT, USAGE, SIGN, and VARYING.

E.19.2 Input distribution

This stage activates any DESTINATION clauses that are defined for the data item to store items in their target locations where indicated. By virtue of the default values, target locations always receive valid data, unless DEFAULT NONE is specified when they are unchanged.

The clauses relevant to this stage are: DESTINATION and VARYING.

E.19.3 Content validation

This stage checks that each data item lies in the expected set of permissible values. 88-level entries may be used to check the values of the data item as a whole and the CLASS clause to check each of its characters. The range of values specified in 88-level entries can depend on specified conditions.

The clauses relevant to this stage are: CLASS, VALUE and VARYING.

E.19.4 Relation validation

This stage checks that data items have appropriate values in relation to any other data items. Other terms for this stage are "inter-field checks" or "cross-field validation". The contents of a data item may be considered to be invalid, depending upon the truth value of a specified condition. For example, the clause 'ITEM-A INVALID WHEN ITEM-B > 35' means that the contents of ITEM-A are always considered to be invalid when the value of ITEM-B is greater than 35.

The clauses relevant to this stage are: INVALID and VARYING.

E.19.5 Error indication

This final stage operates on any VALIDATE-STATUS clauses that are defined and uses them to set up messages or indicators, specified by the programmer, to identify which items, if any, have been rejected. It is also possible to distinguish between the three reasons for rejection. The VALIDATE-STATUS clauses are not placed within the data item being validated but elsewhere in the data division.

The clauses relevant to this stage are: VALIDATE-STATUS and VARYING.

E.19.6 Validation of more complex formats

The data description can contain subordinate entries that have an OCCURS clause. Each repetition of the data item can then be checked independently, can have an independent DESTINATION, and can be assigned an independent error message or indicator. If the data description has alternate formats, these can be described using the REDEFINES clause and the appropriate description can be selected using the PRESENT WHEN clause.

The clauses relevant to these formats are: PRESENT WHEN, REDEFINES, OCCURS, and VARYING.

E.19.7 Examples of validation

The comments placed in these examples explain the use and effects of the various clauses.

Sample 19: Example of validation of USAGE DISPLAY items

```
01 INPUT-RECORD.
*>PIC 99 checks that IN-TYPE is 2 characters numeric;
   03 IN-TYPE PIC 99
*>if IN-TYPE fails the PICTURE check, it is assumed to be 1;
*>       without a DEFAULT clause, the assumed value would here be 0.
      DEFAULT 1.
*>PRESENT WHEN states the condition for this format to be used.
   03 IN-REC-FORMAT-1 PRESENT WHEN IN-TYPE = 0 OR 1 OR 2.
*>PICTURE A(20) checks for 20 alphabetic (or space) characters.
      05 IN-NAME PIC A(20)
*>PRESENT WHEN defines when the validation clauses for this data item apply:
         PRESENT WHEN IN-TYPE = 0 OR 1
*>CLASS checks each character for a class defined in SPECIAL-NAMES.
         CLASS IS ALPHA-UPPER
*>DESTINATION moves this item (or spaces if not alpha) to OUT-NAME.
         DESTINATION OUT-NAME.
      05 FILLER REDEFINES IN-NAME
*>PRESENT WHEN checks whether the item is "blank" under this condition
         PRESENT WHEN IN-TYPE = 2
         DESTINATION OUT-NAME.
            88 VALUE SPACES IS VALID.
*>The values of IN-WEEK are checked to be in non-descending order.
      05 IN-WEEK PIC 99 OCCURS 5
         VARYING IN-WEEK-NO FROM 1, IN-NEXT-WEEK-NO FROM 2
         INVALID WHEN IN-WEEK-NO < 5
         AND IN-WEEK (IN-WEEK-NO) > IN-WEEK (IN-NEXT-WEEK-NO)
*>OUT-WEEK (1) to (5) will hold the values of IN-WEEK (1) to (5),
*> or zero for any one that failed the format (PICTURE) test.
            DESTINATION OUT-WEEK (IN-WEEK-NO).
*>The 88-level INVALID entries check for invalid ranges of values.
         88 VALUES 0, 53 THRU 99 ARE INVALID.
*>REDEFINES and another PRESENT WHEN define an alternate format.
   03 IN-REC-FORMAT-2 REDEFINES IN-REC-FORMAT-1
      PRESENT WHEN IN-TYPE > 2.
*>IN-PAY has insertion characters that must be present on input.
      05 IN-PAY PIC ZZ,ZZZ.ZZ.
*>The 88-level VALID entries check for valid ranges of values;
*> the condition-name, if present, may be used in the usual way.
*> The following assume that DECIMAL POINT IS COMMA is not specified.
         88 IN-PAY-OK VALUES "10,000.00" THRU "20,000.00" ARE VALID.
*>88-level entries may also have a condition attached.
         88 VALUES "20,000.01" THRU "30,000.00" ARE VALID
            WHEN IN-TYPE = 8.
*>exceptional cases can be specified using PRESENT WHEN
      05 IN-CODE PIC AX(3)9(4)
         PRESENT WHEN IN-CODE NOT = "UNKNOWN".
      05 FILLER PIC X(13).
*>
*>*************************************************************
*>Description of target record
*>*************************************************************
*>This is set up by the optional DESTINATION clauses defined
*>in the input record;
*>if a format error is found, a default value is stored instead.
01 TARGET-AREA.
```

```
   05 OUT-NAME PIC X(20).
   05 OUT-WEEK PIC 99 COMP OCCURS 5.
*>
*>*************************************************************
*> Description of error messages
*>*************************************************************
*>Error messages or flags are set up or cleared automatically
*>when the VALIDATE statement is executed; the programmer chooses
*>where they go and what messages or values they contain;
*>they need not be contiguous as they are in this example.
01 VALIDATE-MESSAGES.
   03 PIC X(40) VALIDATE-STATUS "Unknown Record Type - 1 assumed"
      WHEN ERROR FOR IN-TYPE
*> more than one VALIDATE-STATUS clause may be defined in one entry;
*> a NO ERROR phrase produces a message when the item is valid.
      VALIDATE-STATUS "Record type Accepted"
      WHEN NO ERROR FOR IN-TYPE.
*> The VALIDATE-STATUS clause can pinpoint the stage of the failed check.
   03 PIC X(40) VALIDATE-STATUS "Name not alphabetic"
      WHEN ERROR ON FORMAT FOR IN-NAME
      VALIDATE-STATUS "Lower-case not allowed in name"
      WHEN ERROR ON CONTENT FOR IN-NAME
      VALIDATE-STATUS "Name not allowed in this case"
      WHEN ERROR ON RELATION FOR IN-NAME.
*> If no message is stored, spaces will be stored in these cases.
*> Errors may also be indicated by flags;
*> they may also refer to a table of input items.
   03 W-ERROR-FLAG PIC 9 COMP OCCURS 5
      VALIDATE-STATUS 1 WHEN ERROR FOR IN-WEEK.
*>An EC-VALIDATE (non-fatal) exception is also set if the
*>VALIDATE statement detects an invalid condition.
*>
*>*************************************************************
*>Execution of the VALIDATE statement
*>*************************************************************
PROCEDURE DIVISION.
...
*>A single VALIDATE statement performs all the actions implied
*>in the above data descriptions.

   VALIDATE INPUT-RECORD

*>After this statement has been executed:
*>(1) the input record is unchanged;
*>(2) input items are moved automatically to the target area;
*>(3) error messages are set up wherever specified in the program.
```

Appendix D: Introduction to Common exception processing

The information at this site has been excerpted from the "Concepts" Annex of the ISO 2002 COBOL Standard. The section on ***Common exception processing*** begins on page 761.

E.15 Common exception processing

Exception processing is a method for detecting and processing exceptions that occur during the execution of COBOL statements.

There is more than one method of exception processing. The classical methods are: specifying exception phrases on various statements, such as AT END, INVALID KEY, ON EXCEPTION, and so on; checking status values using FILE STATUS; and invoking USE statements based on i-o status codes, open modes, and file-names. All of the classical methods are always enabled, take precedence over common exception processing, and work in exactly the same way they have in previous COBOL standards. The other method of exception processing is called common exception processing. It is based on exception conditions and can be enabled and disabled, depending on a compiler directive. Common exception processing is described in the following paragraphs.

An exception may be due to an error or due to some condition arising during the processing of a statement. When an exception occurs, the associated exception condition exists. When an exception condition exists, further processing takes place as described in the following paragraphs.

Associated with each exception condition is an exception-name or an exception object. The concepts of exception objects are given in E.17.9, Exception objects. The following refers to the predefined exception conditions represented by exception-names. Syntax for processing exception conditions uses these exception-names. They may be specified only in the TURN compiler directive, the RAISING phrase of the EXIT or GOBACK statement, the RAISE statement, and the USE statement. There are three levels of exception-names. Level-1 is one all-inclusive exception name, EC-ALL. Level-2 identifies exceptions associated with a specific type of exception. The level-2 exception-names are: EC-ARGUMENT, EC-BOUND, EC-DATA, EC-FLOW, EC-I-O, EC-IMP, EC-LOCALE, EC-OO, EC-ORDER, EC-OVERFLOW, EC-PROGRAM, EC-RAISING, EC-RANGE, EC-REPORT, EC-SCREEN, EC-SIZE, EC-SORT-MERGE, EC-STORAGE, EC-USER, and EC-VALIDATE. The level-3 exception-names are the level-2 exception-names suffixed by a descriptive character-string that identifies that actual exception condition. When a level-3 exception condition is raised, the associated level-2 and EC-ALL can be used to process the exception, if desired.

The user can define exceptions by suffixing EC-USER-. For example, EC-USER-OVERDRAWN might mean that an account is overdrawn. By executing 'RAISE EC-USER-OVERDRAWN' the user could cause a declarative to be executed.

In a similar fashion, the implementor can suffix EC-IMP- to define exceptions. In this case the implementor defines the fatality, what causes the exception and so on.

Checking for exceptions is initially disabled for all exception conditions, and can be enabled at compile time with the use of the TURN compiler directive. If checking for an exception condition is enabled, it can also be disabled by the TURN compiler directive. The TURN

compiler directive may specify the level-3 exception-name, the associated level-2 exception-name, or EC-ALL.

Checking for an exception condition is locally disabled by the presence of an explicit phrase. For example, if the SIZE ERROR phrase is specified on an arithmetic statement, the raising of the EC-SIZE exception condition is disabled for that statement, except during item identification.

When an exception condition exists and checking for that exception condition is enabled, the exception condition is raised and the last exception status is set to indicate that exception condition. Subsequent processing depends on the presence of explicit phrases or an applicable exception declarative. If there is an explicit phrase, that phrase takes precedence over any declarative or default action. For example, when an arithmetic size error occurs, if an explicit SIZE ERROR or NOT SIZE ERROR phrase is specified, that phrase takes precedence and the EC-SIZE exception condition is not raised. If there is no such phrase, any declarative specified with USE AFTER EXCEPTION EC-SIZE would be executed. The SIZE ERROR and NOT SIZE ERROR phrases will function even if checking for EC-SIZE is not enabled.

If an exception declarative is executed, there are several ways to terminate execution of that declarative. The user can execute 'EXIT ... RAISING' or 'GOBACK RAISING' to cause the exception that caused the declarative to be executed (RAISING LAST EXCEPTION) or another exception ('RAISING EXCEPTION exception-name' or 'RAISING identifier') to be propagated to the activating function, method, or program. This causes that exception condition to exist in the activating runtime element. In addition, the user can execute a RESUME statement to cause execution to be continued at the statement following the statement that caused the exception to be executed (RESUME AT NEXT STATEMENT) or a RESUME statement to cause execution to be continued at a procedure-name that is in the non-declarative portion of the source element (RESUME AT procedure-name). The final method is to fall through the last procedure in the declarative. When this is done, if the exception condition is a fatal exception condition the execution of the run unit is terminated. If the exception condition is not fatal, execution continues as if RESUME AT NEXT STATEMENT were executed.

The default action taken when an exception condition is enabled, the exception exists, no applicable declarative exists, and no explicit phrase is specified on the statement depends on the specific exception condition and other factors. If the exception condition is defined to be non-fatal, execution continues as specified in the rules for the statement. For example, EC-I-O-AT-END will cause execution to continue at the next executable statement following the READ statement. If the exception condition is defined to be fatal, further processing depends on the PROPAGATE compiler directive. If PROPAGATE ON is in effect, execution of the current runtime element is terminated and the exception that occurred is propagated to the activating runtime element, as if an EXIT statement or GOBACK statement with the RAISING LAST EXCEPTION phrase were specified. If PROPAGATE ON is not in effect, execution of the run unit is terminated.

The user can cause an exception to be raised by executing the RAISE statement, primarily for user-defined exceptions.

Additional information about an exception condition is obtained through the use of a series of functions. These functions return detailed information about the last exception status. The functions and their returned values are:

EXCEPTION-FILE returns an alphanumeric character string that contains information about the last I-O status value and any file connector that was associated with the last exception status.

EXCEPTION-FILE-N returns a national character string that contains information about the last I-O status value and any file connector that was associated with the last exception status.

EXCEPTION-LOCATION returns an alphanumeric character string that indicates the location of the statement in which the exception condition associated with the last exception status was raised. Part of the string is implementor-defined.

EXCEPTION-LOCATION-N returns a national character string that indicates the location of the statement in which the exception condition associated with the last exception status was raised. Part of the string is implementor-defined.

EXCEPTION-STATEMENT returns the name of the statement in which the exception condition associated with the last exception status was raised.

EXCEPTION-STATUS returns the exception-name associated with the last exception status.

Exception-names and exception conditions

Table 1: Exception-names and exception conditions, is a list of the exception-names and their attributes.

The meaning of the columns in the table are:

Exception-name - The exception-name associated with an exception condition or a hierarchy of exception-names.

Cat - The category of the exception condition: non-fatal (NF), fatal (Fatal), or implementor-defined (Imp). The category of a level-1 or level-2 exception condition is that of the level-3 exception condition that was raised.

Description - A brief description of what the exception condition means.

Exception-name	Cat	Description
EC-ALL		Any exception
EC-ARGUMENT		Argument error
EC-ARGUMENT-FUNCTION	Fatal	Function argument error
EC-ARGUMENT-IMP	Imp	Implementor-defined argument error
EC-BOUND		Boundary violation
EC-BOUND-IMP	Imp	Implementor-defined boundary violation
EC-BOUND-ODO	Fatal	OCCURS ... DEPENDING ON data item out of bounds
EC-BOUND-PTR	Fatal	Data-pointer contains an address that is out of bounds
EC-BOUND-REF-MOD	Fatal	Reference modifier out of bounds
EC-BOUND-SUBSCRIPT	Fatal	Subscript out of bounds
EC-DATA		Data exception
EC-DATA-CONVERSION	NF	Conversion failed because of incomplete character correspondence
EC-DATA-IMP	Imp	Implementor-defined data exception

Exception-name	Cat	Description
EC-DATA-INCOMPATIBLE	Fatal	Incompatible data exception
EC-DATA-PTR-NULL	Fatal	Based item data-pointer is set to NULL when referenced
EC-FLOW		Execution control flow violation
EC-FLOW-GLOBAL-EXIT	Fatal	EXIT PROGRAM in a global Declarative
EC-FLOW-GLOBAL-GOBACK	Fatal	GOBACK in a global declarative
EC-FLOW-IMP	Imp	Implementor-defined control flow violation
EC-FLOW-RELEASE	Fatal	RELEASE not in range of SORT
EC-FLOW-REPORT	Fatal	GENERATE, INITIATE, or TERMINATE during USE BEFORE REPORTING declarative
EC-FLOW-RETURN	Fatal	RETURN not in range of MERGE or SORT
EC-FLOW-USE	Fatal	A USE statement caused another to be executed
EC-I-O		Input-output exception
EC-I-O-AT-END	NF	I-O status "1x"
EC-I-O-EOP	NF	An end of page condition occurred
EC-I-O-EOP-OVERFLOW	NF	A page overflow condition occurred
EC-I-O-FILE-SHARING	NF	I-O status "6x"
EC-I-O-IMP	Imp	I-O status "9x"
EC-I-O-INVALID-KEY	NF	I-O status "2x"
EC-I-O-LINAGE	Fatal	The value of a LINAGE data-item is not within the required range
EC-I-O-LOGIC-ERROR	Fatal	I-O status "4x"
EC-I-O-PERMANENT-ERROR	Fatal	I-O status "3x"
EC-I-O-RECORD-OPERATION	NF	I-O status "5x"
EC-IMP		Implementor-defined exception condition
EC-IMP-*suffix* (implementor specifies *suffix*)	Imp	Level-3 implementor-defined exception condition
EC-LOCALE		Any locale related exception
EC-LOCALE-IMP	Imp	Implementor-defined locale related exception
EC-LOCALE-INCOMPATIBLE	Fatal	The referenced locale does not specify the expected characters in LC_COLLATE
EC-LOCALE-INVALID	Fatal	Locale content is invalid or incomplete
EC-LOCALE-INVALID-PTR	Fatal	Pointer does not reference a saved locale
EC-LOCALE-MISSING	Fatal	The specified locale is not available
EC-LOCALE-SIZE	Fatal	Digits were truncated in locale editing
EC-OO		Any predefined OO related exception
EC-OO-CONFORMANCE	Fatal	Failure for an object-view
EC-OO-EXCEPTION	Fatal	An exception object was not handled

Exception-name	Cat	Description
EC-OO-IMP	Imp	Implementor-defined OO exception
EC-OO-METHOD	Fatal	Requested method is not available
EC-OO-NULL	Fatal	Method invocation was attempted with a null object reference
EC-OO-RESOURCE	Fatal	Insufficient system resources to create the object or expand the object
EC-OO-UNIVERSAL	Fatal	A runtime type check failed
EC-ORDER		Ordering exception
EC-ORDER-IMP	Imp	Implementor-defined ordering exception
EC-ORDER-NOT-SUPPORTED	Fatal	ISO/IEC 14651:2001 ordering table or ordering level not supported
EC-OVERFLOW		Overflow condition
EC-OVERFLOW-IMP	Imp	Implementor-defined overflow condition
EC-OVERFLOW-STRING	NF	STRING overflow condition
EC-OVERFLOW-UNSTRING	NF	UNSTRING overflow condition
EC-PROGRAM		Inter-program communication exception
EC-PROGRAM-ARG-MISMATCH	Fatal	Parameter mismatch
EC-PROGRAM-ARG-OMITTED	Fatal	A reference to an omitted argument
EC-PROGRAM-CANCEL-ACTIVE	Fatal	Canceled program active
EC-PROGRAM-IMP	Imp	Implementor-defined inter-program communication exception
EC-PROGRAM-NOT-FOUND	Fatal	Called program not found
EC-PROGRAM-PTR-NULL	Fatal	Program-pointer used in CALL is set to NULL
EC-PROGRAM-RECURSIVE-CALL	Fatal	Called program active
EC-PROGRAM-RESOURCES	Fatal	Resources not available for called program
EC-RAISING		EXIT ... RAISING or GOBACK RAISING exception
EC-RAISING-IMP	Imp	Implementor-defined EXIT ... RAISING or GOBACK RAISING exception
EC-RAISING-NOT-SPECIFIED	Fatal	EXIT ... RAISING or GOBACK RAISING an EC-IMP or EC-USER exception condition not specified in RAISING phrase of procedure division header
EC-RANGE		Range exception
EC-RANGE-IMP	Imp	Implementor-defined range exception
EC-RANGE-INDEX	Fatal	Index made negative or too large for container
EC-RANGE-INSPECT-SIZE	Fatal	Size of replace items in INSPECT differs
EC-RANGE-INVALID	NF	Starting value of THROUGH range greater than ending value
EC-RANGE-PERFORM-VARYING	Fatal	Setting of varied item in PERFORM is negative

Exception-name	Cat	Description
EC-RANGE-PTR	Fatal	Pointer SET UP or DOWN is outside range
EC-RANGE-SEARCH-INDEX	NF	No table entry found in SEARCH because initial index out of range
EC-RANGE-SEARCH-NO-MATCH	NF	No table entry found in SEARCH because no entry matched criteria
EC-REPORT		Report writer exception
EC-REPORT-ACTIVE	Fatal	INITIATE on an active report
EC-REPORT-COLUMN-OVERLAP	NF	Overlapping report items
EC-REPORT-FILE-MODE	Fatal	An INITIATE statement was executed for a file connector that was not open in the extend or output mode
EC-REPORT-IMP	Imp	Implementor-defined report writer exception
EC-REPORT-INACTIVE	Fatal	GENERATE or TERMINATE on an inactive report
EC-REPORT-LINE-OVERLAP	NF	Overlapping report lines
EC-REPORT-NOT-TERMINATED	NF	Report file closed with active report
EC-REPORT-PAGE-LIMIT	NF	Vertical page limit exceeded
EC-REPORT-PAGE-WIDTH	NF	Page width exceeded
EC-REPORT-SUM-SIZE	Fatal	Overflow of sum counter
EC-REPORT-VARYING	Fatal	VARYING clause expression non-integer
EC-SCREEN		Screen handling exception
EC-SCREEN-FIELD-OVERLAP	NF	Screen fields overlap
EC-SCREEN-IMP	Imp	Implementor-defined screen handling exception
EC-SCREEN-ITEM-TRUNCATED	NF	Screen field too long for line
EC-SCREEN-LINE-NUMBER	NF	Screen item line number exceeds terminal size
EC-SCREEN-STARTING-COLUMN	NF	Screen item starting column exceeds line size
EC-SIZE		Size error exception
EC-SIZE-ADDRESS	Fatal	Invalid pointer arithmetic
EC-SIZE-EXPONENTIATION	Fatal	Exponentiation rules violated
EC-SIZE-IMP	Imp	Implementor-defined size error exception
EC-SIZE-OVERFLOW	Fatal	Arithmetic overflow in calculation
EC-SIZE-TRUNCATION	Fatal	Significant digits truncated in store
EC-SIZE-UNDERFLOW	Fatal	Floating-point underflow
EC-SIZE-ZERO-DIVIDE	Fatal	Division by zero
EC-SORT-MERGE		SORT or MERGE exception
EC-SORT-MERGE-ACTIVE	Fatal	File SORT or MERGE executed when one is already active
EC-SORT-MERGE-FILE-OPEN	Fatal	A USING or GIVING file is open upon execution of a SORT or MERGE

Exception-name	Cat	Description
EC-SORT-MERGE-IMP	Imp	Implementor-defined SORT or MERGE exception
EC-SORT-MERGE-RELEASE	Fatal	RELEASE record too long or too short
EC-SORT-MERGE-RETURN	Fatal	RETURN executed when at end condition exists
EC-SORT-MERGE-SEQUENCE	Fatal	Sequence error on MERGE USING file
EC-STORAGE		Storage allocation exception
EC-STORAGE-IMP	Imp	Implementor-defined storage allocation exception
EC-STORAGE-NOT-ALLOC	NF	The data-pointer specified in a FREE statement does not identify currently allocated storage
EC-STORAGE-NOT-AVAIL	NF	The amount of storage requested by an ALLOCATE statement is not available
EC-USER		User-defined exception condition
EC-USER-*suffix* (user specifies *suffix*)	NF	Level-3 user-defined exception condition
EC-VALIDATE		VALIDATE exception
EC-VALIDATE-CONTENT	NF	VALIDATE content error
EC-VALIDATE-FORMAT	NF	VALIDATE format error
EC-VALIDATE-IMP	Imp	Implementor-defined VALIDATE exception
EC-VALIDATE-RELATION	NF	VALIDATE relation error
EC-VALIDATE-VARYING	Fatal	VARYING clause expression non-integer

Appendix E: Introduction to Standard arithmetic

The information at this site has been excerpted from the "Concepts" Annex of the ISO 2002 COBOL Standard. The section on ***Standard arithmetic*** begins on page 763.

E.16 Standard arithmetic

When standard arithmetic is in effect most common arithmetic operations will produce results that are predictable, reasonable, and portable. In this context, portable means that the results will be identical from implementation to implementation.

In order to achieve these results, several features are included in COBOL that support standard arithmetic. They include the following:

1) The ARITHMETIC clause in the identification division determines the mode of arithmetic for the source unit.
2) Standard intermediate data items are used to contain each operand in an arithmetic expression and the result of every arithmetic operation, arithmetic expression, and integer and numeric intrinsic functions. A standard intermediate data item is a conceptual data item that the implementor implements as appropriate.
3) The binary arithmetic operators +, –, *, and / and the SQRT function are defined to give results that are accurate to 31 digits. Exponentiation is defined to give results that are accurate to 31 digits for exponents with the values of -4, -3, –2, –1, 0, 1, 2, 3, and 4.
4) The sequence of evaluation of arithmetic expressions is defined.
5) All arithmetic statements, the SUM clause, and many integer and numeric functions are defined in terms of arithmetic expressions. The arithmetic expressions are composed of standard intermediate data items discussed in 2) above and arithmetic operators discussed in 3) above.
6) A size error condition is raised and the EC-SIZE-OVERFLOW exception condition is set to exist whenever the result of any single operation that is part of an arithmetic expression cannot be contained in a standard intermediate data item.
7) The size error condition occurs and the EC-SIZE-OVERFLOW condition exists on certain moves to a resultant identifier and division and exponentiation errors.

The following major decisions are incorporated in standard arithmetic. Specific rationale is outside the scope of this document, but the general considerations include importance of functionality to the using community, cost of execution, and results that are reasonable and expected.

1) The standard intermediate data item contains 32 significant digits.
2) Zero is a unique value of a standard intermediate data item.
3) A standard intermediate data item is described in the form of a floating-point decimal number with a normalized fraction, although the representation may be in any form that provides the same results.

4) The exponent of a standard intermediate data item can range from 999 through -999, inclusive.
5) A result that is close to zero but is not representable in a standard intermediate data item causes a size error condition to be raised and an EC-SIZE-UNDERFLOW exception condition to be set to exist instead of being rounded to zero.
6) In each binary operation and the SQRT function, digits beyond the thirty-second significant digit are truncated. Truncation is a form of rounding that is also referred to as "rounding to zero."
7) Before a compare involving one or more standard intermediate data items, the result is rounded to 31 digits.
8) In exponentiation, even if the base is portable only exponent values of –4, –3, ..., 3, 4 produce portable results.

When standard arithmetic is specified, regardless of whether the results are portable, the results exhibit the following two types of predictability.

1) Within a single execution of a runtime element, arithmetic statements, arithmetic expressions, the SUM clause, and numeric and integer intrinsic functions will yield the same arithmetic results, so long as the value and order of the operands or arguments are the same.
2) Equivalent methods of expressing certain arithmetic produce the same arithmetic result. This is true when the equivalent methods are defined in terms of arithmetic expressions. For example, the following three statements all yield the same arithmetic result:

```
ADD a b c GIVING d
COMPUTE d = FUNCTION SUM (a b c)
COMPUTE d = a + b + c
```

Under standard arithmetic the first two are explicitly defined as being equivalent to the arithmetic expression (a + b + c) which is the same expression given in the third example.

It is important to note that the order of operands and operators can be significant. This is especially true when there is a great difference in the magnitude of the operands in an expression. For example the following two statements may yield different results:

```
ADD a b c GIVING d
ADD c b a GIVING d
```

Another point to be noted is that when standard arithmetic is specified, if any part of an arithmetic expression is implementor-defined then the remainder of the arithmetic expression is still evaluated according to the applicable rules. For example, in the expression

```
13 + A ** 0.4
```

whatever the implementor-defined value of A ** 0.4 is, the result of the entire expression will be exactly the result of adding 13 to that value according to the rules for evaluating arithmetic expressions. If the value of A ** 0.4 is 103.698974 then the value of the full expression will be 116.698974.

E.16.1 Maximum error due to truncation and rounding

This discussion presents the maximum error due to truncation of low-order digits and rounding for primitive operations involving one or two operands. The abbreviation "ulp" means unit in the last position. In general, analysis of a specific situation may reduce the maximum error expected from truncation or rounding; or may eliminate it completely.

E.16.1.1 Arithmetic statements

For the COMPUTE statement, see the paragraph on arithmetic operations in arithmetic expressions below.

E.16.1.1.1 ADD and SUBTRACT statements

Given the following examples:

```
ADD A TO B
ADD C TO D GIVING E
SUBTRACT A FROM B
SUBTRACT C FROM D GIVING E
```

if the composite of operands, the pair A and B or the pair C and D, has fewer than 32 digits and if the destination, B and E, respectively, has at least as many places to the right of the decimal point as the composite of operands, then there is no error due to truncation or rounding. In all other cases, if ROUNDED is specified, the maximum error due to rounding is 1/2 ulp of the destination and if ROUNDED is not specified the maximum error due to truncation is 1 ulp of the destination.

E.16.1.1.2 MULTIPLY statement

Given the following examples:

```
MULTIPLY A BY B
MULTIPLY C BY D GIVING E
```

if the sum of the number of digits in the source operands, the pair A and B or the pair C and D, is less than 32 and if the destination, B and E, respectively, has at least as many places to the right of the decimal point as the sum of the digits to the right of the decimal point of the source operands, then there is no error due to truncation or rounding. In all other cases, if ROUNDED is specified, the maximum error due to rounding is 1/2 ulp of the destination and if ROUNDED is not specified the maximum error due to truncation is 1 ulp of the destination.

E.16.1.1.3 DIVIDE statement

Given the following examples:

```
DIVIDE A INTO C
DIVIDE A INTO B GIVING C
DIVIDE B BY A GIVING C
DIVIDE A INTO B GIVING C REMAINDER D
DIVIDE B BY A GIVING C REMAINDER D
```

the maximum error due to truncation or rounding of the quotient C or remainder D can exceed 1 ulp of the destination for both the quotient and the remainder.

E.16.1.2 Arithmetic operations in arithmetic expressions

In general, the maximum error due to truncation for each primitive operation is 1 ulp of the standard intermediate data item. However, analysis of specific situations may identify cases where the maximum error is less or there is no such error. Rounding does not apply.

E.16.1.2.1 Unary operations and operands with no operator

Given the following examples:

```
+A
–A
A
```

there is no loss due to truncation.

E.16.1.2.2 Addition and subtraction

Given the following examples:

```
A + B
A – B
```

If the composite of operands A and B has fewer than 33 digits, there is no loss due to truncation.

E.16.1.2.3 Multiplication

Given the following example:

```
A * B
```

If the sum of the number of significant digits in A and B is fewer than 33, there is no loss due to truncation.

E.16.1.2.4 Division

Given the following example:

```
A / B
```

There is no way to predict a reduced error from truncation by examining the number of digits in either the dividend A or the divisor B.

E.16.1.2.5 Exponentiation

Given the following example:

```
A ** B
```

There is no loss due to truncation in the following cases:

- B has a value of zero;
- B has a value of one;
- B has a value of two and A has fewer than 17 significant digits;
- B has a value of three and A has fewer than 11 significant digits;
- B has a value of four and A has fewer than 9 significant digits.

E.16.1.2.6 SQRT function

Given the following example:

```
Function SQRT (A)
```

There is no way to predict a reduced error from truncation by examining the number of digits.

E.16.2 Examples

These examples demonstrate a few of the features and requirements of standard arithmetic. In order to make the examples easier to follow, the maximum number of digits in a data division data item is assumed to be 4 and the standard intermediate data item contains 5 significant digits. The results before truncation and normalization are only one possible format and do not necessarily correspond to any real implementation, however the algebraic value after truncation is what is required.

In the following examples, "sidi" stands for "standard intermediate data item" and "ir-n" stands for the "nth intermediate result". The notation used for the standard intermediate data items and intermediate results is scientific notation. For example "+1.10000 E+4" and "+ .11000 E+5" are both representations of eleven thousand and "+ .06999 E+5" and "+ .69990 E+4" both represent six thousand nine hundred ninety-nine.

```
1 A   pic s9(4) value +8000.
1 B   pic s9(4) value +3000.
1 C   pic s9(4) value –4001.
1 D   pic s9(4) value is zero.
1 E   pic s99v99 value +56.79
```

Sample 20: COMPUTE D = A + B + C.

```
begin
   calculate ir-1 = A + B
      convert A to sidi                                          +8.0000 E+3
      convert B to sidi                                          +3.0000 E+3
      add (one possible representation)                         +11.0000 E+3
      normalize and truncate, store in ir-1                      +1.1000 E+4
   calculate ir-2 = ir-1 + C
      no conversion needed for ir-1                              +1.1000 E+4
      convert C to sidi                                          –4.0010 E+3
      add (one possible representation)                          +0.6999 E+4
      normalize (truncation not needed), store in ir-2           +6.9990 E+3
   store ir-2 in D
      round from last                                            +6.9990 E+3
      move, D =                                                        +6999
end
```

Sample 21: COMPUTE D ROUNDED = D + E

```
begin
   calculate ir-1 = D + E
      convert D to sidi                                          +6.9990 E+3
      convert E to sidi                                          +5.6790 E+1
```

```
      add (one possible representation)                          +7.05579 E+3
      (normalize not needed) truncate, store in ir-1              +7.0557 E+3
   store ir-1 in D
      (Because of ROUNDED, the sidi is not rounded.
      However, since D has the maximum number of
      digits, the effect is the same.)
      apply ROUNDED
         align                                                       + 7055.7
         round causes increase in ulp                                  + 7056
         move, D =                                                     + 7056
   end



1 E   pic S9(4) value +1291.
1 F   pic S9(3) value +569.
```

Sample 22: COMPUTE E = E * F

```
begin
calculate ir-1 = E * F
   convert E to sidi                                           +1.2910 E+3
   convert F to sidi                                           +5.6900 E+2
   multiply (one possible representation)                     +0.734579 E+6
   normalize, truncate, store in ir-1                          +7.3457 E+5
store ir-1 in E
   round from last                                              +7.346 E+5
   move, SIZE ERROR
end
```

Sample 23: COMPUTE E = F * 0.754

```
begin
calculate ir-1 = F * 0.754
   convert F to sidi                                           +5.6900 E+2
   convert 0.754 to sidi                                       +7.5400 E-1
   multiply (one possible representation)                     +4.29026 E+2
   (normalize not needed) truncate, store in ir-1              +4.2902 E+2
store ir-1 in F
   round from last                                              +4.290 E+2
   move, E =                                                        +429
end
```