SHARE in San Antonio 2016

SHARE EDUCATE • NETWORK • INFLUENCE

Impacting the future of the enterprise technology ecosystem

Enterprise COBOL V6.1: What's New?

Tom Ross 'Captain COBOL'

February 29

What new features are in Enterprise COBOL V6?

- Improved compiler capacity to allow compilation and optimization of very large COBOL programs
- COBOL 2002: ALLOCATE and FREE statements
- COBOL 2002: INITIALIZE … TO VALUE and more!
- JSON GENERATE statement
- New and modified compiler options
- COBOL 2002: VSAM "0x" file status code current 97
- Improved memory requirements of TABLE SORT and performance improvements

Improved Compiler Capacity

- COBOL 5.x is unable to compile very large programs
  - o Analysis shows that the backend runs out of memory
  - o Backend performs memory and compute intensive operations such as program analysis, optimization and code-generation
  - o Optimization engine scales back when it sees that it is running low on available memory at OPT(1|2)
  - o However there are certain programs which do not compile even at OPT(0)

Improved Compiler Capacity

- COBOL V6.1 has an improved version of the code generator/optimizer
  - o Compile time will be somewhat slower due to costs of AMODE 64
  - o Backend invokes certain COBOL runtime library routines during compilation
  - o Since the COBOL runtime is AMODE 31 there will be some overhead of marshalling and un-marshalling of data passed between the AMODE 64 backend and AMODE 31 runtime.
  - o Future release of the product will ship with a compatible runtime and recover from this compile-time degradation
- We think this is a reasonable short term trade off to get all customer code compiling

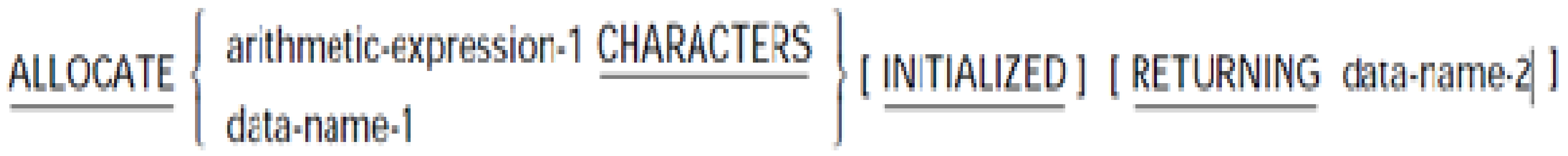
Improved Compiler Capacity

- Compiler is now sensitive to z/OS MEMLIMIT setting
  - In Enterprise COBOL V6, the compiler may start using storage above the 2GB BAR to compile very large programs.
  - This means that the z/OS MEMLIMIT parameter would have to be set to a non-zero value. The z/OS default for MEMLIMIT is 2GB.
  - How much to increase MEMLIMIT depends on many factors, such as OPT option level, size of program, complexity of program, and if certain language features are repeated. Some large programs can compile with a small amount memory, and some smaller programs might need a lot. The indication that you do not have enough (for instance, if your MEMLIMT setting is 0) is the compiler error message:

    IGYCB7145-U Insufficient memory in the compiler to continue compilation.

ALLOCATE statement

The ALLOCATE statement obtains dynamic storage.
Syntax:



```
ALLOCATE { arithmetic-expression-1 CHARACTERS } [ INITIALIZED ] [ RETURNING data-name-2 ]
         { data-name-1                        }
```

- The data item referenced by data-name-1 shall be an 01 or 77 LINKAGE SECTION data item.
- Data-name-2 reference a data item of USAGE POINTER.
- If DATA(24) option is specified the storage will be obtained below the 16M line.

ALLOCATE statement

- ALLOCATE … INITIALIZED
- If both the INITIALIZED phrase and arithmetic-expression-1 are specified, all bytes of the allocated storage are initialized to binary zeros.
- If both the INITIALIZED phrase and data-name-1 are specified, the allocated storage is initialized as if an INITIALIZE data-name-1 WITH FILLER ALL TO VALUE THEN TO DEFAULT statement were executed.
  – Note: This implies that a VALUE clause on a LINKAGE SECTION item is no longer just a comment!

ALLOCATE statement

Examples:

```
  01 ptr1 USAGE POINTER.
  01 var1 PIC X(500).
LINKAGE SECTION.
  01 dn1 pic x(1000).

  ALLOCATE dn1
  ALLOCATE dn1 INITIALIZED RETURNING ptr1.

  ALLOCATE 1000 CHARACTERS RETURNING ptr1.
  ALLOCATE length of var1 CHARACTERS RETURNING ptr1.

  ALLOCATE function length (var1) +200 CHARACTERS
                      RETURNING ptr1.
```

FREE statement

The FREE statement releases dynamic storage previously obtained with an ALLOCATE statement.

Syntax:

FREE { data-name-1 } …

The data item referenced by data-name-1 shall be of USAGE POINTER.

FREE statement

Example:

```
01 ptr1 USAGE POINTER.

ALLOCATE 1000 CHARACTERS RETURNING ptr1.
.
.
.
FREE ptr1.
```

INITIALIZE statement

The INITIALIZE statement sets selected categories of data fields to predetermined values. The INITIALIZE statement is functionally equivalent to one or more MOVE statements.

INITIALIZE { identifier-1 } ... [WITH FILLER]

[{ ALL | category-name } TO VALUE]

[THEN REPLACING { category-name DATA BY { identifier-2 | literal-1} } …]

[THEN TO DEFAULT]

- Where category-name is:
  - ❑ ALPHABETIC
  - ❑ ALPHANUMERIC
  - ❑ ALPHANUMERIC-EDITED
  - ❑ NATIONAL
  - ❑ NATIONAL-EDITED
  - ❑ NUMERIC
  - ❑ NUMERIC-EDITED
  - ❑ DBCS
  - ❑ EGCS

INITIALIZE statement

INITIALIZE { identifier-1 } ... [WITH FILLER]
[{ ALL | category-name } TO VALUE]
[THEN REPLACING { category-name DATA BY { identifier-2 | literal-1} } …]
[THEN TO DEFAULT]

6 a. If the data item qualifies as a receiving-operand because of the VALUE phrase:

– The sending-operand is determined by the literal in the VALUE clause specified in the data description entry of the data item.
  If the data item is a table element, the literal in the VALUE clause that corresponds to the occurrence being initialized determines the sending-operand.
  The actual sending-operand is a literal that, when moved to the receiving-operand with a MOVE statement, produces the same result as the initial value of the data item as produced by the application of the VALUE clause.

INITIALIZE statement

INITIALIZE { identifier-1 } ... [WITH FILLER]
 [{ ALL | category-name } TO VALUE]
[THEN REPLACING { category-name DATA BY { identifier-2 | literal-1} } …]
[THEN TO DEFAULT]

6 b. If the data item does not qualify as a receiving-operand because of the VALUE phrase, but does qualify because of the REPLACING phrase, the sending-operand is the literal-1 or identifier-2 associated with the category specified in the REPLACING phrase.

6 c. If the data item does not qualify in accordance with general rules 6a and 6b, the sending-operand used depends on the category of the receiving-operand as follows:

INITIALIZE statement

Receiving operand	Figurative constant
Alphabetic	SPACES
Alphanumeric	SPACES
Alphanumeric-edited	SPACES
National	SPACES
National-edited	SPACES
Numeric	ZEROES
Numeric-edited	ZEROES

JSON GENERATE

JSON GENERATE *identifier-1* FROM *identifier-2*
   [COUNT [IN] *identifier-3*]
   [NAME [OF] {*identifier-4* [IS] *literal 1*}...
   [SUPPRESS {*identifier-5*}...]
   [[ON] EXCEPTION *imperative-statement-1*]
   [NOT [ON] EXCEPTION *imperative-statement-2*]
[END-JSON]

- Very similar to XML GENERATE

JSON GENERATE

Example of output….

JSON GENERATE json-text FROM G

```
 01 G.
     05 h.
        10 a pic x(10)    Value ‘Eh?’.
        10 3_ pic 9         Value 5.
        10 C-c pic x(10) Value ‘See’.
```

```
{“G": {“h": {“a": "Eh?", "3_": 5, “C-c": "See"}}}
```

New and modified compiler options

- The new VSAMOPENFS(COMPAT|SUCC) option allows you to change File Status=97 into File Status=00 for certain VSAM OPEN statements.
- The new SUPPRESS|NOSUPRESS option enables or disables the SUPPRESS phrase of COPY statements.
  - Longtime SHARE user requirement!

New and modified compiler options

- The new SSRANGE(ZLEN|NOZLEN) suboption allows a 0-length reference modification.
  - Code checks for length >= 0 with ZLEN
  - Code checks for length > 0 with NOZLEN, like all previous COBOL compilers
  - Quite a few users could not use SSRANGE because of this zero-length reference modification restriction
  - EGL (Visual Gen) can now use SSRANGE(ZLEN) !
- The diagnostic message for the ZONECHECK(MSG) compiler option is improved by adding the data item contents for the offending data item and also adding the program name of the program that contained the offending data item.

New and modified compiler options

- The LVLINFO installation option has been removed and replaced by a 7-character build-level identifier, of the format PYYMMDD, that is added to the compiler listing header.
- The build-level identifier is placed in locations that previously held the following LVLINFO data:
  - Listing header
  - Signature information bytes
  - ADATA field called PTF Level

FILE STATUS 97

- This file status applies to VSAM during OPEN
  - The file was successfully opened, even though the returned status code was non-zero in previous COBOL releases
  - The new VSAMOPENFS(SUCC) compiler option allows you to change File Status=97 into File Status=00 for these VSAM OPEN statements
  - The new VSAMOPENFS(COMPAT) compiler option keeps compatible behavior of File Status=97 for OPEN statements that have “File integrity verified”

Table Sort Improvements and Performance Improvements

- The Table Sort feature was introduced in COBOL V5R2. The COBOL library routine doing the actual sorting currently requires a fair amount of storage as work area. Improvements to reduce storage requirements of this routine were made.
  - Some performance tuning was also done
- Performance tuning on INSPECT, UNSTRING and SEARCH ALL (when searching large tables).
  - Input to INSPECT and UNSTRING can be very long for some programs, especially those that involve XML processing. Tuning was done on theses library routines so that performance can scale up to very large input

Product-related enhancements

- ❑ Enterprise COBOL for z/OS, V6.1 delivers the following runtime and performance-related enhancements:
  - o WORKING-STORAGE will be acquired from HEAP storage in all cases, so that there are (almost) no exceptions to when the STORAGE(xx) runtime option will affect WORKING-STORAGE.
    - ✓ SPANNED files are the exception
  - o In the previous version, this was only true for CICS and some special non-CICS cases.
- ❑ Reduced storage requirements and performance tuning is implemented in Table SORT.
- ❑ Performance improvements are implemented for INSPECT, UNSTRING, and SEARCH ALL statements

QUESTIONS?