IBM Enterprise COBOL for z/OS 6.5
What's New

Feb 25, 2026
SHARE Orlando
Tom Ross

IBM

Agenda

- Major features
  - IBM z17 support
  - VSAMDB support
  - User-defined types
  - Improved Integration with ABO and Z Code Optimization Advice – new SMARTBIN suboptions
- Miscellaneous changes
  - z/OS versions supported
  - IBM Z hardware supported
  - ARCH/TUNE changes
  - OMITTED class test and OPTIONAL parameters changes
  - COBOL/Java interoperability changes
  - COBOL Runtime updates

z17 Support

Enterprise COBOL 6.5 supports the latest features of IBM z17 to maximize your hardware investment, reduce CPU usage, and improve performance of critical COBOL applications.

IBM z17 support – ARCH(15) / TUNE(15)

ARCH(15)

- Produces code that uses instructions available on the 9175-xxxx (IBM z17) model in IBM z/Architecture mode
- Specifically, the ARCH(15) machine and its follow-ons add instructions supported by the new **Vector Enhancement Facility 3** and **Vector Packed Decimal Enhancement Facility 3**
    - The Vector Enhancement Facility 3 adds performance improvements for COBOL programs that contain numeric calculations involving COMP/COMP-5 binary data types that require more than 64 bits
    - The Vector Packed Decimal Enhancement Facility 3 adds performance improvements for COBOL programs that are compiled with the NUMCHECK option

TUNE(15)

- Generates code that is optimized for the 9175-xxx (IBM z17) model in IBM z/Architecture mode

See ARCH and TUNE options

IBM z17 support – Performance improvements

Performance Tuning Guide

- The Performance Tuning Guide has been updated with measurements for Enterprise COBOL 6.5 on an IBM z17 system

See https://www.ibm.com/docs/en/cobol-zos/6.5.0?topic=performance-tuning-guide

Please take note of the first topic, Performance Measurements, of the Performance Tuning Guide:

- The performance measurements in this information were made with COBOL 6.5 on a z17 system except where otherwise noted. The programs used were batch-type (noninteractive) applications. Unless otherwise indicated, all performance comparisons that are made in this information are referencing CPU time performance and not elapsed time performance.
- **Note:** Except where otherwise noted, performance comparisons in this document are measured using microbenchmarks. Each microbenchmark is designed to highlight the compiler's performance improvement in a specific area. Microbenchmarks are made to amplify the performance of instructions generated for a specific COBOL statement and help the compiler team choose the optimal set of instructions for a specific COBOL statement. They are not indicative of the overall performance of a real-world application and should not be used as a measure or indicator of the overall performance of a real-world COBOL application. Real programs typically use a mix of features. A program will only see a performance improvement from improvements in the compiler if the program spends a significant percentage of its time executing code that is affected by those improvements.
- **Note**: Performance results for customer applications will vary, depending on the source code, the compiler options specified, and other factors

VSAMDB

Support for a modern, unstructured NoSQL database (VSAMDB).

VSAMDB files (datasets) provide the capability of storing and retrieving JSON documents through VSAM in COBOL. For more information, refer to Processing VSAMDB files.

This function is compatible with IBM EzNoSQL for z/OS

VSAMDB ...

- Is a NoSQL document database on z/OS.
- Is built to store JSON or BSON documents
- Contains semi-structured documents with flexible schemas.
    - Each document may have a unique structure.
- Allows for rapid modern application development.
- Is scalable.
- Is highly available.

VSAMDB: System Requirements - APARs

The VSAMDB feature in this requires various APARs to enable the latest features of VSAMDB.

z/OS 2.5 or higher is required with the following APARs/PTFs:

- OA66674, VSAMDB Miscellaneous Fixes
- OA65272, which PRE-reqs the following APARs:

  OA64279, OA64335, OA64482, OA64523, OA64660, OA64854, OA64239, OA63987, OA64954, OA65898, OA65711, OA65027, OA66342, OA66970, OA65323, OA67062, OA67038

See the following page for System Requirements

VSAMDB: System Requirements

VSAMDB also has specific hardware and software requirements.

VSAMDB is accessed by VSAM Record Level Sharing (RLS). VSAMDB files inherit the full SYSPLEX data sharing capabilities currently provided to other VSAM RLS applications. Thus a dependency on an IBM Parallel Sysplex configuration is required. For information on how to configure a Sysplex refer to: https://www.ibm.com/docs/en/zos/3.1.0?topic=mvs-zos-setting-up-sysplex

VSAMDB: System Requirements

The minimum hardware/software configuration requires:

1. At least one logical partition (LPAR) running IBM's Version 2.5 z/OS or above in plex mode with the above mentioned APARs.
2. At least one internal or external Coupling Facility (CF) attached to the LPAR(s) (see PR/SM Planning Guide: https://www.ibm.com/support/pages/sites/default/files/inline-files/SB10-7175-01a.pdf)
3. Enable the SMSVSAM address space on each LPAR which will access the VSAMDB datasets
4. Configure one or more SMS storage classes (STORCLAS) containing a CACHESET (see Defining storage classes for VSAM RLS: https://www.ibm.com/docs/en/zos/3.1.0?topic=sharing-defining-storage-classes-vsam-rls#dscrls). The CACHESET identifies the name of one or more CF cache structures for use by the VSAMDB databases (see Defining VSAM RLS attributes in data classes: https://www.ibm.com/docs/en/zos/3.1.0?topic=sharing-defining-vsam-rls-attributes-in-data-classes).

Additional information on how to define and access a VSAM RLS Database (VSAMDB) can be found at: https://www.ibm.com/docs/en/zos/3.1.0?topic=sharing-defining-accessing-vsam-rls-database in the Z/OS DFSMS Using Data Sets documentation.

COBOL VSAMDB Support: To support reading, writing, updating, and deleting of JSON documents in a VSAMDB file(database) using COBOL’s file processing statements. Compatible with IBM EzNoSQL for z/OS data.

VSAMDB Background Information

VSAMDB

Refers to a VSAM Key Sequenced Data Set (KSDS) defined with the DATABASE(JSON) or DATABASE(BSON) option (IDCAMS).

Payload

VSAMDB files store documents (records) containing JavaScript Object Notation (JSON) or Binary JSON (BSON).

JSON

JSON is a data interchange format (for computer-to-computer messaging) encoded as UTF-8 text.
See json.org. Example object containing a JSON key/value pair:

{ “message”: “Hello World” }

APIs

Current APIs to use VSAMDB files include z/OS Assembler Macros, and the “EzNoSQL” C, Python and Java libraries.

Purpose of VSAMDB

To provide an accessible and searchable repository of un-structured JSON or BSON documents.

More VSAMDB Background Information

VSAMDB Keys

VSAMDB keys are identified by a name – they are not positional. Keys can be located at arbitrary positions within the JSON document payload.

Key Values

VSAMDB key values are JSON values. They can be strings, numbers, objects, etc.

Example JSON:
{ “first-name”:“John”,
“last-name”:”Doe” }
Key is “first-name”
Key value is “John”

Length of key values

Key values have varying lengths from document to document.

Primary/Alternate keys

VSAMDB supports both primary and alternate keys. If a primary key is not defined explicitly, and implicit primary key is defined named “znsq_id”.

Ordered/Unordered Keys

Primary Keys may be ordered or unordered. Unordered keys cannot be retrieved in ordered sequence but provide performance benefits during record insertion. Alternate keys are ordered.

Uniqueness

Primary keys must be unique. Alternate keys can be unique or non-unique.

Example JCL snippet

IDCAMS definition of a VSAMDB file.

(Not shown: Alternate key indexes and Path datasets – see docs in “z/OS DFSMS Using Data Sets” for examples.

```
DEFINE CLUSTER (NAME(JSON.KSDS)    -    !  Base
cluster:  JSON.KSDS
        STORCLAS(storclasname)     -    !
        LOG(NONE)                  -    !
Recoverability
        DATABASE(JSON)             -    !  JSON
documents to be stored
        KEYNAME(first-name)        -    !  Primary
key
```

Example JSON documents

Example 1:
Primary key is “first-name”.

Primary key value for this document is “John”.

Primary keys must have a unique value.

Keys can be anywhere within document but keys appearing first in document reduce parsing.

Example 2: Nested keys
Primary key is “first” within the “name” object.

IDCAMS definition would be:

DATABASE(JSON)
KEYNAME(name\first)

```
{ “first-name”:”John”,
  “last-name”:”Doe”,
  “address”:{
    “street”:”123 Fifth Avenue”,
    “city”:”New York City”,
    “state-abbr”:”NY”,
    “country-code”:”US”
  } }
```

```
{ “name”: {
    “first”:”John”,
    “last”:”Doe” }
  “address”:{
    “street”:”123 Fifth Avenue”,
    “city”:”New York City”,
    “state-abbr”:”NY”,
    “country-code”:”US”
  } }
```

VSAMDB: COBOL Support

1. File I/O (Examples following on next slides)
    1. "VSAMDB-" SELECT clause DD Name prefix enables VSAMDB file I/O.
    2. Record keys must be PIC U BYTE-LENGTH items.
    3. LENGTH phrase of RECORD KEY clause must specify key length in bytes.
    4. File records should be defined as RECORD IS VARYING DEPENDING ON <length-item> with a PIC U BYTE-LENGTH item as the record.
    5. <length-item> contains the record length in bytes.
2. Auxiliary language enhancements for UTF-8 string handing.
    1. A new phrase COUNT BYTES <item> in the JSON GENERATE phrase indicates the number of bytes generated in the output.
    2. The ULENGTH intrinsic function (that returns the counted number of UTF-8 code points) has been enhanced with two additional, optional, parameters to specify a starting byte position and byte length: FUNCTION ULENGTH(data start-pos length)

VSAMDB: COBOL Program

1) Use new VSAMDB- prefix on DD name.

2) Use new LENGTH phrase on RECORD KEY clause to indicate a satellite item that contains byte length for the key.

3) Use DEPENDING ON phrase to indicate record length in bytes.

4) Define file record with PIC U BYTE-LENGTH or PIC U DYNAMIC.

5) Record key is NOT defined in file record. Key can be in W-S, L-S, LINKAGE sections. Use PIC U BYTE-LENGTH or PIC U DYNAMIC.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     EXAMPLE1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE1 ASSIGN TO VSAMDB-INDD0
    ORGANIZATION IS INDEXED
    ACCESS MODE IS RANDOM
    RECORD KEY IS VSAMDB-PKEY
      LENGTH IS VSAMDB-PKEY-LEN BYTES.

DATA DIVISION.
FILE SECTION.
FD  FILE1    RECORD IS VARYING IN SIZE
      DEPENDING ON FILE1-REC-LEN.
01 FILE1-REC-PAYLOAD PIC U BYTE-LENGTH 640.

WORKING-STORAGE SECTION.
01 FILE1-REC-LEN PIC 9(9) BINARY.
01 VSAMDB-PKEY PIC U BYTE-LENGTH 240.
01 VSAMDB-PKEY-LEN PIC 9(9) BINARY.
01 USER-DATA.
  05 FIRST-NAME PIC U(20).
  05 LAST-NAME PIC U(20).
  05 ADDRESS.
    10 STREET PIC U(20).
  ...
```

VSAMDB: COBOL Program Direct WRITE Example

1) Initialize user data.

2) Generate the file payload. COUNT BYTES is a new phrase returning the number of bytes (rather than UTF-8 characters/code-points ala COUNT) of the generated JSON. Note FILE1-REC-LENGTH from previous slide is specified on the FD.

3) Open file for writing new records.

4) Direct (indexed) write of record to file. Underneath the covers VSAMDB parses JSON for keys and updates indexes.

```
PROCEDURE DIVISION.

MOVE "John" TO FIRST-NAME
MOVE "Doe" TO LAST-NAME
…

JSON GENERATE FILE1-REC-PAYLOAD FROM USER-DATA
  COUNT BYTES IN FILE1-REC-LENGTH
  NAME USER-DATA IS OMITTED
       FIRST-NAME IS "first-name"
       LAST-NAME IS "last-name"
END-JSON

* JSON will look like:
* {"first-name":"John","last-name":"Doe" … }

OPEN OUTPUT FILE1

WRITE FILE1-REC-PAYLOAD

CLOSE FILE1
```

VSAMDB: How to get key value after a WRITE to a keyless base.

If a VSAMDB file has been constructed without a primary key an implicit unique 122 byte primary key is generated and inserted (by DFSMS) automatically into each document written to the database. The key is generated with key name "znsq_id". Example:

"znsq_id":"a1b2c3d4...."

For convenience, COBOL will automatically populate the primary key item (specified on the RECORD KEY clause) with the key value upon execution of the WRITE statement. Users can then use the JSON PARSE statement to parse the key value (which is in JSON format) into a COBOL data item.

Note: the primary key item must be specified as PIC U BYTE-LENGTH 122 or larger.

VSAMDB: COBOL Program Direct READ Example

1) Generate desired key value as JSON. Note VSAMDB-PKEY is the RECORD KEY and VSAMDB-PKEY-LEN is specified as key length (in bytes) on RECORD KEY clause. The result is the 6 UTF-8 character string “John” (including the quotes).

2) Direct (indexed) read using primary key.

3) (Optional) Parse JSON into COBOL data.

```
PROCEDURE DIVISION.

MOVE “John” TO FIRST-NAME
JSON GENERATE VSAMDB-PKEY FROM
FIRST-NAME
  COUNT BYTES IN VSAMDB-PKEY-LEN
  NAME FIRST-NAME IS OMITTED
* Generated JSON looks like:
* “John”

OPEN INPUT FILE1
READ FILE1 KEY IS VSAMDB-PKEY

JSON PARSE FILE1-REC-PAYLOAD INTO
USER-DATA
  NAME USER-DATA IS OMITTED
       FIRST-NAME IS “first-name”
       LAST-NAME IS “last-name”
       …
END-JSON

CLOSE FILE1
```

VSAMDB: Other supported ACCESS modes.

Supported access modes include:

ACCESS MODE	Description
ACCESS IS RANDOM	Access documents using a primary or alternate key.
ACCESS IS SEQUENTIAL	Access records sequentially via ascending order or unordered (depending on order of key)
ACCESS IS DYNAMIC	Both by index and sequentially – for example by searching for a document by key (using the START statement) and then iterating over documents sequentially.

The START statement (cursor positioning) will include support for:

- START KEY = (exact key match)
- START KEY >= (Greater than or equal to key matching)
- (new keyword) START KEY IS = PARTIAL phrase – match part of the key.

Example of partial key match:
Programmer wants to retrieve documents where the key value, a telephone number, starts with “555-”. They can use a partial key match with key value “555-” using START KEY IS EQUAL PARTIAL to position to the first of those documents and iterate sequentially using READ NEXT.

VSAMDB: COBOL Program Sequential READ Example

1) User wants to read records whose FIRST-NAME starts with "J". Generate the key value using JSON GENERATE. The result is the 3 UTF-8 character string “J” (including the quotes).

(Note: ACCESS MODE IS SEQUENTIAL or DYNAMIC must be specified in FILE CONTROL for sequential reading)

2) Position file cursor.

3) Sequential READ.

4) (Optional) Parse JSON into COBOL data.

```
PROCEDURE DIVISION.

MOVE “J” TO FIRST-NAME
JSON GENERATE VSAMDB-PKEY FROM FIRST-NAME
  COUNT BYTES IN VSAMDB-PKEY-LEN
  NAME FIRST-NAME IS OMITTED
* Generated JSON looks like:
* “J”

OPEN INPUT FILE1
START KEY IS EQUAL PARTIAL VSAMDB-PKEY
READ FILE1

JSON PARSE FILE1-REC-PAYLOAD INTO USER-
DATA
  NAME USER-DATA IS OMITTED
       FIRST-NAME IS “first-name”
       LAST-NAME IS “last-name”
       …
END-JSON

CLOSE FILE1
```

Some more VSAMDB definitions

Non-recoverable datasets
- Do not participate in transactional recovery.
- Use the IDCAMS LOG(NONE) parameter.

Recoverable Datasets
- May participate in transactional recovery.
- Must have DFSMStvs (transactional VSAM) configured and enabled
- Use the IDCAMS LOG(UNDO) or LOG(ALL) parameter.
- The COBOL application should add commit points via the SRRCMIT or use TVS autocommit feature

Read Integrity: 3 Levels
1. NRI – No Read Integrity. VSAM allows the application to read all records but there is no guarantee on integrity.
2. CR – Consistent Read. VSAM will obtain a SHARE lock on each record requested by the application for the duration of the read; lock is released automatically after.
3. CRE – Consistent Read Extended. The application starts a transaction during which VSAM obtains SHARE locks for records accessed and holds them for the duration of the transaction. A commit or rollback must be performed (by the application) to release locks. Requires DFSMStvs.

Read Integrity can be specified as JCL parameter:
DD ddname RLS=[NRI|CR|CRE]

JSON GENERATE COUNT BYTES

Use the new COUNT BYTES phrase to retrieve the number of generated bytes – unlike the COUNT phrase which retrieves the number of generated UTF-8 code points (characters).

This is particularly useful when specifying key lengths and record lengths for VSAMDB files, which use byte lengths.

```
PROCEDURE DIVISION.
...
   JSON GENERATE JSON-TEXT FROM COBOL-DATA
      COUNT IN CHAR-COUNT
      COUNT BYTES IN BYTE-COUNT
...
```

New ULENGTH optional parameters

The ULENGTH intrinsic function now supports two additional optional parameters. The first optional parameter specifies the starting byte position, while the second specifies the byte length.

This enhancement is useful when users want to calculate the number of UTF-8 code points contained in a portion of a fixed length UTF-8 item.

```
PROCEDURE DIVISION.
...
   COMPUTE CHAR-COUNT = FUNCTION
   ULENGTH(UTF8-TEXT BYTEPOS,
           BYTELEN)
...
```

VSAMDB:
Extended Examples

1. How to define VSAMDB datasets
2. COBOL VSAMDB sample programs
3. How to compile and link the COBOL program
4. How to RUN the COBOL program with VSAMDB datasets

1. How to define VSAMDB datasets

VSAMDB datasets run on a SYSPLEX that has RLS enabled.

Minimum configuration is one LPAR and one internal or external CF.

See member IGYVDB1J in the SIGYSAMP dataset for full JCL.

The JCL builds a VSAMDB dataset with the following properties.

1. primary index keyname field called "REC-ID" (KEYNAME)
2. records are saved in key order (ORDEREDINDEX)
3. Using STORAGE CLASS (OS390) as defined during sysplex/RLS setup (STORCLAS)

2. COBOL VSAMDB Sample Programs

The SIGYSAMP dataset contains the following sample programs that use VSAMDB:

- IGYVDB1A – Main program that calls two sub-programs:
  - IGYVDB1B – Initializes a (traditional) VSAM file with data
  - IGYVDB1C – Reads the VSAM file, converts the data into JSON (using JSON GENERATE), and stores the JSON into a VSAMDB file
- IGYVDB1J – JCL for IGYVDB1A/B/C that:
  - Allocates VSAM and VSAMDB files
  - Compiles, links, runs IGYVDB1A/B/C.
- IGYVDB2* – Similar to IGYVDB1* but instead initializes a VSAMDB file, reads the JSON from the file, converts the JSON into COBOL data (using JSON PARSE) and writes the COBOL data to a (traditional) VSAM file.
- IGYVDB3* - Uses both a primary and alternate key, with fixed length keys and records.
- IGYVDB4* - Uses both a primary and alternate key, with dynamic length keys and records.
- IGYVDB5* - Uses an implicit primary key, with dynamic length keys and records.

3. How to compile and link the COBOL program

No special options are required to read VSAMDB files.

The FILE-CONTROL information and the FILE SECTION must use the VSAMDB syntax to identify the access as VSAMDB.

Compile and link the program as you would normally.

Note: LP(64) is not supported for VSAMDB

Example JCL to compile and link the COBOL PROGRAM

```
----------------------------------------------
//COBOL1   EXEC PGM=IGYCRCTL,
// REGION=200M,
// PARM=' OPTFILE'
//STEPLIB  DD
DSN=TSC390.COBOL.IGY.V6R5M0.GOOD.SIGYCOMP,DISP=SHR
//        DD DSN=TSCTEST.CEEZ250.SCEERUN,DISP=SHR
//        DD DSN=TSCTEST.CEEZ250.SCEERUN2,DISP=SHR
//SYSIN    DD DSN=&&SOURCE(TEST001),DISP=(SHR,PASS)
//SYSLIN   DD DSN=&&OBJECT(TEST001),DISP=(SHR,PASS)
//SYSOPTF  DD DATA,DLM='/>'

/>
//SYSMDECK DD UNIT=SYSALLDA,SPACE=(CYL,(10,10))
//SYSPRINT DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
```

3. How to compile and link the COBOL program

No special options are required to read VSAMDB files.

The FILE-CONTROL information and the FILE SECTION must use the VSAMDB syntax to identify the access as VSAMDB.

Compile and link the program as you would normally.

Example JCL to compile and link the COBOL PROGRAM

```
----------------------------------------------
//*************************************************************
//* LINK
//*************************************************************
//SYS//LKED1    EXEC PGM=IEWL,
// PARM='LIST,MAP  '
//SYSLIB   DD DSN=TSCTEST.CEEZ250.SCEELKEX,DISP=SHR
//        DD DSN=TSCTEST.CEEZ250.SCEELKED,DISP=SHR
//        DD DSN=TSCTEST.CEEZ250.SCEELIB,DISP=SHR
//OBJECT   DD DSN=&&OBJECT,DISP=(SHR,PASS)
//SYSLIN   DD DATA,DLM='/>'
   INCLUDE OBJECT(TEST001)
/>
//SYSLMOD  DD DSN=&&LOAD(TEST001),DISP=(SHR,PASS)
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSOUT   DD DUMMY
//SYSPRINT DD SYSOUT=*
```

4. How to RUN the COBOL program with VSAMDB datasets

The JCL to invoke the linked VSAMDB COBOL example follows the same rules as normal VSAM – the alternate index PATH dataset must be named using the same root name as the primary cluster, appended with a digit.

In the COBOL program, The file-control section identifies the DDNAME for the primary index.
The alternate index must use the same DDNAME suffixed with the next logical alternate index position.
Example:

```
FILE-CONTROL.
   SELECT FILE1 ASSIGN TO VSAMDB-FIL1A1
   ORGANIZATION Indexed
   ACCESS MODE IS dynamic
   RECORD KEY IS VSAMDB-KEY-1
     LENGTH IS VSAMDB-KEY-1-BLEN BYTES
  ALTERNATE RECORD KEY IS  VSAMDB-ALT-1
   LENGTH IS VSAMDB-ALT-1-BLEN BYTES
```

The ddname for the primary INDEX will be : FILE1A1
The ddname for the alternate INDEX will be: FILE1A11

If more alternate indexes exist, they would be : FILE1A12 then FILE1A13 etc.

example GO STEP JCL:

```
--------------------
//GO1     EXEC PGM=TEST001,
// PARM=''
//STEPLIB  DD DSN=&&LOAD,DISP=(SHR,PASS)
//        DD DSN=TSCTEST.CEEZ250.SCEERUN,DISP=SHR
//        DD DSN=TSCTEST.CEEZ250.SCEERUN2,DISP=SHR
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//* LIST the VSAMDBS in FILE-CONTROL order
//FIL1A1  DD DISP=SHR,
// DSN=MYHLQ.GCHFDDDI.BASE.KSDS1
//FIL1A11 DD DISP=SHR,
// DSN=MYHLQ.GCHFDDDI.PATH1
//*******************************************************
```

VSAMDB: Limitations

1. LP(64) is not supported.
2. Writing to Recoverable Datasets is not supported.
3. Read integrity CRE is not supported.

User-defined types

A user-defined type is a group item or elementary item that is defined with the TYPEDEF clause and can be used to create one or more data items (called type instances), whose data description entries and structure are identical to those of the type.

Type instances are defined using the TYPE clause. For more information on type declarations, refer to User-defined types.

User-defined types via TYPEDEF/TYPE keywords

- The COBOL standard includes the TYPEDEF and TYPE keywords to provide full support for user-defined types in COBOL
  - The TYPEDEF keyword is used to introduce the definition of a user-defined type
  - The TYPE keyword is used to define an *instance* of a user-defined type
- There are two flavours of typed data items in COBOL:
  - **Weakly-typed items** (TYPEDEF) **[Supported in 6.5]**
    - A weakly-typed item has all the characteristics of its corresponding TYPEDF but is treated as a mere “shorthand” for the data description entry of the typedef. There is no enforcement of COBOL type checking rules for weakly-typed items during operations such as compare, MOVE and CALL. Weakly typed items can be either group items or elementary items.
  - **Strongly-typed items** (TYPEDEF STRONG) **[Not yet supported]**
    - A strongly-typed group item has all the characteristics of its corresponding TYPEDEF but COBOL type checking rules are enforced for the item on operations such as compare, MOVE and CALL. Strongly-typed items can only be groups according to the COBOL standard.

User-defined type examples

- Example 1: define and instantiate a typed *elementary* item

```
* elementary type definition
 01 MY-STRING-T TYPEDEF PIC X(20)
JUSTIFIED.

* type instance definitions
 01 STR1 TYPE MY-STRING-T.
 01 STR2 TYPE MY-STRING-T.

* define elementary type instance
within a group
 01 G1.
   03 S1 TYPE MY-STRING-T.
 :
     MOVE STR1 TO STR2
     MOVE 'Hello, world!' TO S1
```

User-defined type examples

- Example 2: Define and instantiate a typed *group* item

```
* elementary type definition
 01 MY-STRING-T TYPEDEF PIC X(20) JUSTIFIED.

* group type definition
 01 MY-GROUP-T TYPEDEF.
   03 S1 TYPE MY-STRING-T.
   03 N1 PIC S9(9).

* type instance definitions
 01 G1 TYPE MY-GROUP-T. *> G1 is a group item
 01 G2 TYPE MY-GROUP-T. *> G2 is a group item
 :
     MOVE S1 OF G1 TO S1 OF G2
     COMPUTE N1 OF G1 = (N1 OF G1 + N1 OF G2)
* 2
```

Basic rules: TYPEDEF clause

Syntax:

```
[IS] TYPEDEF
```

- Defines a weak type

- The TYPEDEF clause can only be specified on 01- and 77-level items, but 01-level items that contain the TYPEDEF clause may be elementary items or group items (i.e., an "elementary type" or a "group type")
- If a TYPEDEF clause appears in a data description entry, it must be the first clause after the data name
- A data description entry with the TYPEDEF clause cannot be renamed or redefined
- Neither the EXTERNAL clause nor the VOLATILE clause is compatible with the TYPEDEF clause
- When GLOBAL is used with TYPEDEF it applies to the type itself and not instances of the type (i.e., types are scoped)
- The definition of a type using TYPEDEF must appear in the data division *before* it is referenced (i.e., no forward references)
- The TYPEDEF clause may be used on data description entries in any section of the data division. They may be referenced anywhere in the data division after that, including in different sections
  - NOTE: In the file section, type definitions must appear under an FD or SD but are not specifically associated with that FD or SD

Basic rules: TYPE clause

Syntax:

```
TYPE [TO] type-name
```

- Define an instance of a type
- *type-name* refers to the name of a previously defined type definition that must include the TYPEDEF clause

- The TYPE clause may appear anywhere in a data description entry after the data name
- A data description entry containing the TYPE clause must NOT include the following clauses (these attributes should come from the type itself):
  - PICTURE
  - USAGE
  - SIGN
  - BLANK WHEN ZERO
  - DYNAMIC
  - JUSTIFIED
  - SYNCHRONIZED
  - REDEFINES
- Clauses that **are** allowed with the TYPE clause:
  - OCCURS
  - VALUE
  - EXTERNAL
  - VOLATILE
  - GLOBAL
- *type-name* must have been defined prior to using it in a TYPE clause
- A data description entry with a TYPE clause cannot have any subordinate items (including level-88 items). Any subordinates must be part of the type itself
- A TYPE clause that appears on a 77-level item cannot refer to a group type

Miscellaneous rules

- `VALUE` clauses:
  - When a VALUE clause is specified in a data description entry that includes the TYPE clause, if the corresponding type includes a VALUE clause at that same level, then it will be overridden (VALUE clauses on subordinate items in a group are not affected)

- Types and compilation units (CU):
  - If a type name is referenced in a CU, the definition of that type must also appear somewhere in that CU *before* it is referenced
  - This means that every prototype, user-defined function and external program (each of which is its own CU) must include the definition of a type if the type is referenced somewhere within them, but the definition only needs to appear once in the CU
    - Note: a type defined in a particular data division section (e.g., WORKING-STORAGE section) can be referenced in any subsequent section (e.g., the LINKAGE section)

User-defined types -- more examples

- Example 3: define and instantiate weakly typed items that include additional clauses.
  - Demonstrates how clauses such as VALUE and OCCURS can be used in a data description entry that contains a TYPE clause

```
* elementary type definition
01 MY-STRING-T TYPEDEF PIC X(20) JUSTIFIED
      VALUE 'Init value 0'.

* type instance definitions override VALUE
* clause in type definitions
01 STR1 TYPE MY-STRING-T VALUE 'Init value 1'.
01 STR2 TYPE MY-STRING-T.
01 G1.
  03 STR3 TYPE MY-STRING-T OCCURS 10 TIMES.
:
    DISPLAY 'STR1 = ' STR1 *> 'Init value 1'
    DISPLAY 'STR2 = ' STR2 *> 'Init value 0'
    MOVE 'Hello' TO STR3(5)
```

User-defined types -- more examples

- Example 4: handling level-88 items with elementary types
  - Demonstrates how level-88 items should be used with elementary types

```
* CORRECT WAY:
* Put level-88 items in elementary typedef
 01 MY-ENUM-T TYPEDEF PIC 9(9) BINARY.
  88 VAL1 VALUE 1.
  88 VAL2 VALUE 2.

* type instance definitions
 01 NUM1 TYPE MY-ENUM-T.
:
    SET VAL1 of NUM1 TO TRUE

* WRONG WAY:
* Try level-88 items on elementary type
* instance
  01 MY-ENUM-T TYPEDEF PIC 9(9) BINARY.

* type instance definitions
  01 NUM1 TYPE MY-ENUM-T.
    88 VAL1 VALUE 1.  *> not allowed
    88 VAL2 VALUE 2.  *> not allowed
:
    SET VAL1 OF NUM1 TO TRUE
```

New features supporting TYPEDEF/TYPE

- New high-low qualification operator ::
  - When there are multiples instances of a group type, qualifying subordinate members will be common – maybe there is a better way to do it
  - The new high-low qualification operator :: (an alternative to "OF") allows a streamlined way to qualify subordinate items from highest-level name part to lowest level name part, common in other languages
  - :: can be used to qualify subordinate items in non type-related group items as well

Example:

```
* type definition
01 MYTYPE-T TYPEDEF.
  03 G1.
    05 G2.
      07 DATA1 PIC X(20).
      07 DATA2 PIC S9(9) COMP-3.

* type instance definitions
01 GRP1 TYPE MYTYPE-T.
01 GRP2 TYPE MYTYPE-T.
:
   MOVE 23 TO
   GRP1::G1::G2::DATA2      *> new way
   MOVE 48 TO
   DATA2 OF G2 OF G1 OF GRP1 *> old way
```

New features supporting TYPEDEF/TYPE

- Index name qualification
  - Previously, index names could not be qualified by the name of the table with which they are associated
  - This would be problematic if there are multiple instances of a type that includes a table with index names. There would be ambiguity.
  - Index names can now be qualified like other subordinate items in a group, even for non-type-related items

Example:

```
* type definition
01 MYTYPE-T TYPEDEF.
  03 MYTAB OCCURS 10 TIMES INDEXED BY INX.
    05 DATA1 PIC X(20).
    05 DATA2 PIC S9(9) COMP-3.

* type instance definitions
01 GRP1 TYPE MYTYPE-T.
01 GRP2 TYPE MYTYPE-T.
:
     SET GRP1::INX TO 8
     MOVE 'Hello, world 1!' TO
          GRP1::DATA1(GRP1::INX)
     SET GRP2::INX TO 4
     MOVE 'Hello, world 2!' TO
          GRP2::DATA1(GRP2::INX)
```

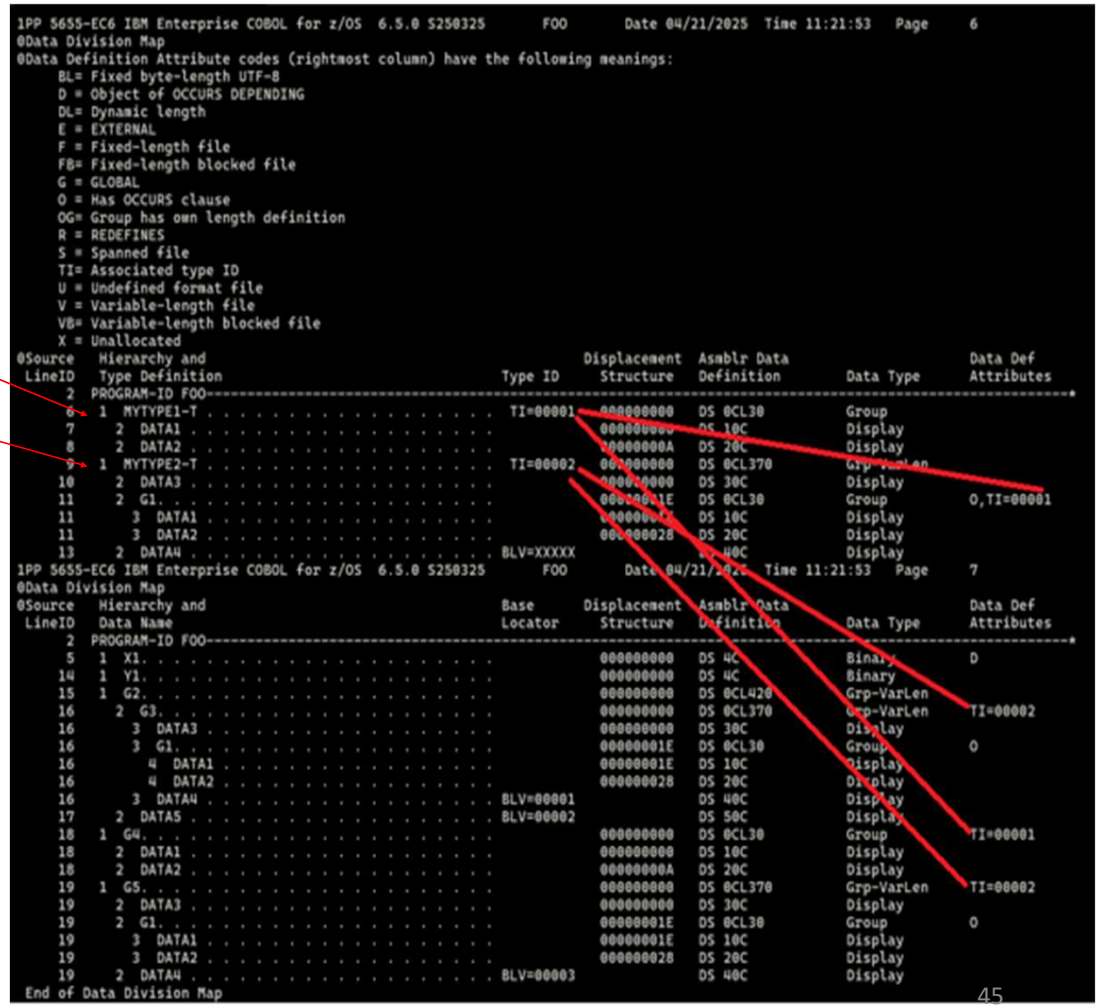
New features supporting TYPEDEF/TYPE

- When the MAP option is in effect, there will be a new map section for type definitions

- Only types that are referenced in data item definitions in the data division will be included in the map
- Types are assigned unique IDs
- Type instances, both in the type section of the map and the data name section of the map, will refer to the type definition in the type section using the unique type ID
- Types will appear in expanded form - i.e., type instances in a type definition will be expanded out to show the full form of the type when it is instantiated; however, the corresponding type ID will be provided for reference

Type map listing example

```
    identification division.
        program-id. foo.
        data division.
        working-storage section.
        01 x1 pic s9(9) binary.
        01 mytype1-t typedef.
          03 data1 pic x(10).
          03 data2 pic x(20).
        01 mytype2-t typedef.
          03 data3 pic x(30).
          03 g1 type mytype1-t occurs 1
to 10 times
                depending on x1.
          03 data4 pic x(40).
        01 y1 pic s9(9) binary.
        01 g2.
          03 g3 type mytype2-t.
          03 data5 pic x(50).
        01 g4 type mytype1-t.
        01 g5 type mytype2-t.
        procedure division.
            ….
```



```
1PP 5655-EC6 IBM Enterprise COBOL for z/OS  6.5.0 S250325       FOO       Date 04/21/2025  Time 11:21:53   Page     6
@Data Division Map
@Data Definition Attribute codes (rightmost column) have the following meanings:
     BL= Fixed byte-length UTF-8
     D = Object of OCCURS DEPENDING
     DL= Dynamic length
     E = EXTERNAL
     F = Fixed-length file
     FB= Fixed-length blocked file
     G = GLOBAL
     O = Has OCCURS clause
     OG= Group has own length definition
     R = REDEFINES
     S = Spanned file
     TI= Associated type ID
     U = Undefined format file
     V = Variable-length file
     VB= Variable-length blocked file
     X = Unallocated
@Source    Hierarchy and                                          Displacement  Asmblr Data                                Data Def
 LineID    Type Definition                              Type ID     Structure    Definition        Data Type               Attributes
      2   PROGRAM-ID FOO----------------------------------------------------------------------------------------------------*
      6    1  MYTYPE1-T . . . . . . . . . . . . . . . . . . . .  TI=00001  00000000   DS 0CL30          Group
      7      2  DATA1 . . . . . . . . . . . . . . . . . . . . .            00000000   DS 10C            Display
      8      2  DATA2 . . . . . . . . . . . . . . . . . . . . .            0000000A   DS 20C            Display
      9    1  MYTYPE2-T . . . . . . . . . . . . . . . . . . . .  TI=00002  00000000   DS 0CL370         Grp-VarLen
     10      2  DATA3 . . . . . . . . . . . . . . . . . . . . .            00000000   DS 30C            Display
     11      2  G1. . . . . . . . . . . . . . . . . . . . . . .            0000001E   DS 0CL30          Group                  0,TI=00001
     11        3  DATA1 . . . . . . . . . . . . . . . . . . . .            0000001E   DS 10C            Display
     11        3  DATA2 . . . . . . . . . . . . . . . . . . . .            00000028   DS 20C            Display
     13      2  DATA4 . . . . . . . . . . . . . . . . . . . . . BLV=XXXXX            DS 40C            Display
1PP 5655-EC6 IBM Enterprise COBOL for z/OS  6.5.0 S250325       FOO       Date 04/21/2025  Time 11:21:53   Page     7
@Data Division Map
@Source    Hierarchy and                                Base     Displacement  Asmblr Data                                Data Def
 LineID    Data Name                                    Locator   Structure    Definition        Data Type               Attributes
      2   PROGRAM-ID FOO----------------------------------------------------------------------------------------------------*
      5    1  X1. . . . . . . . . . . . . . . . . . . . . . .                  00000000   DS 4C             Binary                 D
     14    1  Y1. . . . . . . . . . . . . . . . . . . . . . .                  00000000   DS 4C             Binary
     15    1  G2. . . . . . . . . . . . . . . . . . . . . . .                  00000000   DS 0CL420         Grp-VarLen
     16      2  G3. . . . . . . . . . . . . . . . . . . . . .                  00000000   DS 0CL370         Grp-VarLen             TI=00002
     16        3  DATA3 . . . . . . . . . . . . . . . . . . .                  00000000   DS 30C            Display
     16        3  G1. . . . . . . . . . . . . . . . . . . . .                  0000001E   DS 0CL30          Group                  0
     16          4  DATA1 . . . . . . . . . . . . . . . . . .                  0000001E   DS 10C            Display
     16          4  DATA2 . . . . . . . . . . . . . . . . . .                  00000028   DS 20C            Display
     16        3  DATA4 . . . . . . . . . . . . . . . . . . . BLV=00001        DS 40C            Display
     17      2  DATA5 . . . . . . . . . . . . . . . . . . . . BLV=00002        DS 50C            Display
     18    1  G4. . . . . . . . . . . . . . . . . . . . . . .                  00000000   DS 0CL30          Group                  TI=00001
     18      2  DATA1 . . . . . . . . . . . . . . . . . . . . .                  00000000   DS 10C            Display
     18      2  DATA2 . . . . . . . . . . . . . . . . . . . . .                  0000000A   DS 20C            Display
     19    1  G5. . . . . . . . . . . . . . . . . . . . . . .                  00000000   DS 0CL370         Grp-VarLen             TI=00002
     19      2  DATA3 . . . . . . . . . . . . . . . . . . . . .                  00000000   DS 30C            Display
     19      2  G1. . . . . . . . . . . . . . . . . . . . . . .                  0000001E   DS 0CL30          Group                  0
     19        3  DATA1 . . . . . . . . . . . . . . . . . . . .                  0000001E   DS 10C            Display
     19        3  DATA2 . . . . . . . . . . . . . . . . . . . .                  00000028   DS 20C            Display
     19      2  DATA4 . . . . . . . . . . . . . . . . . . . . BLV=00003        DS 40C            Display
 End of Data Division Map
```

New features supporting TYPEDEF/TYPE

- ADATA support
  - SYSADATA records X'0024', X'0030' and X'0042' have been updated to represent TYPE/TYPEDEF-related information
    - Symbol record: X'0042'
      - New value for "Symbol type" field.
        - X'30' (Type-name)
          - Indicates an item defined with the TYPEDEF clause
          - Group types appear fully expanded in the ADATA
      - New field 'Type-name Symbol ID' (FL4)
        - Immediately follows "Dynamic-length item limit" field
        - set in every type X'0042' record that is part of a type instance and contains the Symbol ID of the corresponding type X'0042' record of a TYPEDEF where the properties of this symbol were derived
      - New value for "Storage type field"
        - 22 (Typedef)
    - Parse tree record: X'0024'
      - New node subtypes
        - 0025 (TYPEDEF)
        - 0026 (TYPE)
    - Token record: X'0030'
      - New token code
        - 0344 (TYPEDEF)

New features supporting TYPEDEF/TYPE

- Debug support via TEST option
- When TEST(DWARF) is in effect, DWARF debug information sufficient for fully debugging type instances is generated
  - For types, this DWARF debug information is largely indistinguishable from the DWARF that would be generated had the resulting item been defined manually without using types
  - NOTE: There is no DWARF info specifically for the types themselves

Type support in 6.5

- Limitations and known bugs

- **Limitation**: An ODO table in a group type cannot have an ODO object that is defined in any type, including the same type as the ODO table itself. However, the ODO object in the type can be a regular data item *outside* a type definition. *This will be a restriction for the initial release but may be relaxed in a future continuous delivery PTF.*
  - *Note diagnostic messaging around this issue still needs improvement*

User-defined types pros/cons

- **Pros:**
  - Allows clean definitions of user-defined data structures – no need to pollute data definitions with special “substitution tags” for use by COPY…REPLACING
  - No need to use the COPY statement to create multiple instances of a data structure
  - Guarantees consistent definitions of structures and consistent use of field names across multiple modules
  - Improves data integrity by supporting type-checking on various operations such as compare, MOVE and CALL (**strongly-typed data items only, not supported in 6.5 GA**)
- **Cons:**
  - Since subordinate items in typed group items will no longer be unique, more explicit name qualification is needed, which can make programs more verbose

Improved integration with ABO and Z Code Optimization Advice

Improved Integration with ABO and Z Code Optimization Advice

- New SMARTBIN(NAMES|NONAMES) suboptions
- Default is:
  - SMARTBIN when LP(32) is in effect, and NOSMARTBIN with LP(64).
    - When SMARTBIN is specified without suboptions, the default changes is SMARTBIN(NONAMES).
  - SMARTBIN is not supported when LP(64) is in effect
- When SMARTBIN(NAMES) is specified, information about the names of the user symbols will be added to the additional binary metadata.
- Compiling with SMARTBIN(NAMES) will improve the analysis output of ***IBM watsonx Code Assistant for Z Code Optimization Advice***.

- SMARTBIN(NAMES) is not required for ABO optimization

- *See SMARTBIN for more information*

Miscellaneous changes

z/OS versions supported

The following z/OS versions are supported with Enterprise COBOL 6.5

- z/OS 2.5
- z/OS 3.1
- z/OS 3.2 (in beta, GAs Sept 2025)

Required APARs/PTFs:

- COBOL RT APARs PH66224 and PH66225 (available in May 2025 COBOL RT PTFs) provide runtime support for Enterprise COBOL 6.5 compiler features.
- PTFs for APARs OA65272 and OA66674 (VSAM EzNOSQL) are required when using the COBOL 6.5 VSAMDB feature.
- These are called out as Mandatory Operation Requisites in the Program Directory, but there is no prereq added to COBOL 6.5 SMP/E package. Sysprogs must install them separately.

IBM z hardware supported

Enterprise COBOL 6.5 and programs that it generates run on the following IBM servers

- IBM z17
- IBM z16
- IBM z15 or IBM z15 T02
- IBM z14 or IBM z14 ZR1
- IBM z13 or IBM z13s

ARCH / TUNE changes

The ARCH option specifies the machine architecture for which the executable program instructions are to be generated.

- ARCH(15) has been added to support the new IBM z17 machines
- ARCH(10) has been removed since IBM System zEC12 machines are no longer supported
- The default ARCH is changed from 10 to 11

The TUNE option specifies the architecture for which the executable program will be optimized.

- TUNE(15) has been added to support the new IBM z17 machines
- TUNE(10) has been removed since IBM System zEC12 machines are no longer supported
- The default TUNE is changed from 10 to 11

See ARCH and TUNE options for more information

OMITTED class test and OPTIONAL parameters changes

- BY REFERENCE parameters in PROCEDURE DIVISION USING statements can now be marked as OPTIONAL.

- Optional parameters can be tested using the new OMITTED class test to find if they were included or omitted in the CALL statement to this program.

- *See Omitted argument condition for more information*

- *These features can be used to extend applications by adding parameters for subprograms without having to change all of the callers!*

- *Examples:*
  PROCEDURE DIVISION USING A B OPTIONAL C

  If C is OMITTED Then
  Display 'The calling program did not pass C'
  Else
  Display 'The calling program passed ' C

COBOL/Java interoperability enhancements in 6.5

- Summary of changes in 6.4 PTFs that are also in 6.5
  - APAR PH61700
    - The "methods file" input to cjbuild has been made optional
      - Instead, all stub files found in the directory indicated by `-c/--coboldir` will be linked into the DLL for the application
      - To take advantage of this, users should organize their code so that the stub files for all files in an interoperable application are in a single directory on z/OS UNIX and *only* the stub files for that application are in the directory
      - Using cjbuild in this way is highly recommended
    - If you choose to continue to use the "methods" file, you no longer have to put the names of individual Java methods called from COBOL into the file. Instead you only need to put the name of the COBOL program making the calls to Java in the file (the name must appear as it is specified in the `program-id` clause (without any quotes))

COBOL/Java interoperability enhancements in 6.5

- APAR61700 (cont)
  - The "methods file" is now referred to as the "program file" in the docs because it no longer contains any Java methods – only the names of COBOL programs
  - We now allow users to specify the programs that they would normally specify in the program file in the cjbuild command invocation itself using the `-i/--progid <program-id>` option, and that option can be specified as many times as needed (once per program)
  - New option –r`/--jar <jar-file-name>` is provided to allow users to create a Java .jar file of all the Java files generated by the compiler and cjbuild for the application

COBOL/Java interoperability enhancements in 6.5

- Example (with/without methods file)
  - Assume application has two COBOL programs: prog1.cbl, prog2.cbl
  - prog1.cbl and prog2.cbl call Java method `enterprise.COBOL.doStuff1()` and `enterprise.COBOL.doStuff2()`
  - Previously you would create a methods file (let's call it "methods") containing:

    Java.enterprise.cobol.doStuff1
    Java.enterprise.cobol.doStuff2

- Building:

```
cob2 prog1.cbl -q"javaiop(outpath('/home/user/jiop/out'))"
cob2 prog2.cbl -q"javaiop(outpath('/home/user/jiop/out'))"
...
cjbuild –v --coboldir /home/user/jiop/out methods app1 # old way
cjbuild –v --coboldir /home/user/jiop/out app1 #new way(no methods file)
```

COBOL/Java interoperability enhancements in 6.5

- If you do want to use a methods file still, you no longer need to add in the names of the Java methods you call individually. Just put the name of the COBOL program making calls
- Alternatively, you could use `-i/--progid` option to specify the program names involved instead of a methods file (cleaner, and more typical for compiler commands)
- For this example, it would be:
  prog1
  prog2
- Build commands:

```
cob2 prog1.cbl -q"javaiop(outpath('/home/user/jiop/out'))"
cob2 prog2.cbl -q"javaiop(outpath('/home/user/jiop/out'))"
...
cjbuild –v --coboldir /home/user/jiop/out methods app1
OR
cjbuild –v --coboldir /home/user/jiop/out --progid prog1 –progid prog2 app1
```

COBOL/Java interoperability enhancements in 6.5

- APAR PH63024
  - New option –k/--cleanscript <clean-script-name>
    - A clean script with the specified name is generated in the directory you run cjbuild in to clean up any artifacts generated by cjbuild
      - Note: the cleanscript is generated unconditionally with default name cleanscript.sh
    - If you specify script option – when running the script, the script will also clean up any artifacts generated by the compiler itself that it finds in the directory

COBOL/Java interoperability change in 6.5

- Non-backwards compatible change to DLL naming convention:

- Starting with 6.5, we have made it so that, by default, the name of the DLL created by cjbuild when the output location **is a data set** is just the DLL "base name" provided to the cjbuild utility via the "`-d/--dlloutdir`" option, **i.e., it no longer contains the prefix "LIB".**
  - This allows users to use all 8 characters of the member name space unlike before where they were limited to 5 chars unnecessarily (the "lib" prefix was only needed on UNIX)
  - Note: When the output location of the DLL is a z/OS UNIX directory, the name provided by the user is still prefixed with the name "lib" as per the normal UNIX convention regarding shared libraries.

- Recommended remediation:
  - When re-building applications, users should update the link step of the build JCL for their programs, which is where the side deck for the DLL built by cjbuild will be mentioned
    - The LIB prefix will need to be removed from the side deck

COBOL/Java interoperability changes in 6.5

- Starting in 6.5, all COBOL stub files produced by the compiler have been combined into a single file
  - e.g., prog1_stub.cbl

  - The goal is that soon (6.5 PTF) we can start to allow users to choose to output the stub file to a data set instead of a USS directory, which would be much more to their liking

- As mentioned previously, it is no longer necessary to put the names of individual Java methods in the methods file (since APAR PH61700, in 6.4)

  - Starting in 6.5, if users include a Java method name in the methods file, they get a warning from cjbuild that it will be ignored

COBOL Runtime - COBOL 6.5 Support

The following PTFs are required for COBOL 6.5

- LE 3.2: z/OS 3.2 GA (includes all COBOL Runtime work up to the July 2025 PTF)
- LE 2.5: May 2025 PTF - 31-bit: UO03154, UO03155
  64-bit: UO03158
- LE 3.1: May 2025 PTF - 31-bit: UO03156, UO03157
  64-bit: UO03159

* If tried without the proper Runtime PTF installed, at runtime, you get the following severe error message IGZ0153S Program X1 was compiled with a level of the compiler that requires service to be installed on z/OS or on the COBOL Runtime Component of the Language Environment, or both. Contact COBOL support for further assistance.

* Refer to 'COBOL 6 Runtime PTFs' on the COBOL Fix list website:
https://www.ibm.com/support/pages/fix-list-and-new-features-enterprise-cobol-zos#rt0425

* COBOL Runtime is always backwards-compatible
That is, the latest/greatest Runtime PTF works with any COBOL programs
Refer to 'Table 2. Runtime libraries for COBOL programs, product identifiers, components, and EOS dates'
https://www.ibm.com/docs/en/cobol-zos/6.4.0?topic=overview-cobol-compiler-versions-required-runtimes-support-information

COBOL Runtime - IPCS support for COBOL 6.5

IPCS VERBEXIT ‘ALL’ uses COBOL Runtime (.MIGLIB dataset) for the COBOL sections.

* No major changes but fixed many bugs
* It might crash when a COBOL 6.5 dump is read and the system is missing the latest COBOL Runtime PTF
* The 2025 June COBOL Runtime PTF ships the up-to-date COBOL RT IPCS work
* Newly added a version info in the IPCS output. Available with the 2025 June COBOL Runtime PTF and afterwards

(For dump listing, the COBOL RT listing has been moved to
https://github.ibm.com/zCompilers/cobrte-PTF-listings/branches/all
Or /home/cobdev/cobolListingWorkspace on torolabW)

COBOL Runtime - IPCS support for COBOL 6.5

```
      +000020 27065F6C - +0009FF 2706694B              same as above

 No PIPICB associated with CAA at address : 26A1C1E8

*********************************************************************************
                              COBOLV5+ ENVIRONMENT DATA
---------------------------------------------------------------------------------
COBOL FORMATTER INFO: PH66706 (2025-06-02 10:23:37 HLQ=TSC390.G.COB65.Z31.N)

   COBEDB:  27030568
   +000000  IDENT:COBEDB       LENGTH:00000184    INPL_MAIN:26A0C238
   +000014  FLAGS:88000000     MAIN_CLLE:270315B8       DUMMY_CLLE:27030E00
   +000020  FREE_CLLE:00000000       CLLE_HASH:27030A00
   +000028  FIRST_CLLE:270315B8      EXTD_LIST:00000000
   +000030  EXT_FILE_BLOCK:00000000        EXT_FILESYNC@:00000000
   +000038  V4_EVENT_NF:26C46500     V4_OTH_EVENT_NF:26CA9490
   +000040  CLLESRCH:26D2F7E8        FUNCDESC:26D30390
   +000048  CLLE_CANCEL:26D222C8     QSAM_EOD_DSP_FUNC:00000000
   +000064  QSAM_GET_DSP_FUNC:00000000     QSAM_PUT_DSP_FUNC:00000000
   +00006C  QSAM_PUTX_DSP_FUNC:00000000    QSAM_UGET_DSP_FUNC:00000000
   +000074  QSAM_UPUT_DSP_FUNC:00000000    QSAM_VPUT_DSP_FUNC:00000000
```

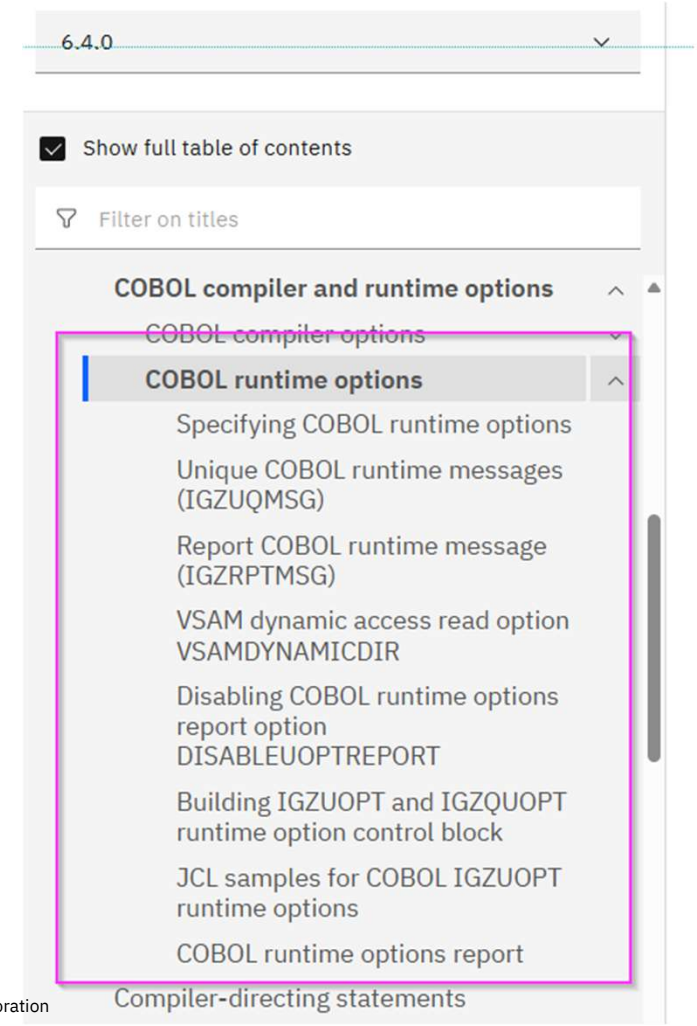
COBOL Runtime - expanding COBOL Runtime specific options

Expanding more of COBOL Runtime specific options, prefixed with IGZ.
Don’t get confused with the BASE LE Runtime options
(RPTOPTS, STACK, STORAGE, …)

For eg, recently added
IGZUQMSG (Unique COBOL runtime message)
IGZRPTMSG (Report COBOL runtime message)
… more coming

Programming Guide:
Compiling and debugging your program
> COBOL runtime options:
https://www.ibm.com/docs/en/cobol-zos/6.4.0?topic=options-cobol-runtime

COBOL Runtime - expanding COBOL Runtime specific options



Resources

- Enterprise COBOL Product Page
  - https://www.ibm.com/products/cobol-compiler-zos
- Enterprise COBOL Documentation Library
  - https://www.ibm.com/support/pages/enterprise-cobol-zos-documentation-library
- Enterprise COBOL for z/OS 6.5 announcements
  - https://www.ibm.com/docs/en/announcements/enterprise-cobol-zos-65-unleashes-power-z17-modernizes-cobol-applications-user-defined-types-vsamdb-support
  - https://www.ibm.com/docs/en/announcements/enterprise-cobol-value-unit-edition-zos-65-unleashes-power-z17-modernizes-your-cobol-applications-user-defined-types-vsamdb-support
- Enterprise COBOL Upgrade Portal
  - https://www.ibm.com/docs/SS6SG3_latest/migration-portal.html
- Enterprise COBOL Support Library
  - https://www.ibm.com/support/pages/ibm-enterprise-cobol-zos-support-library
- COBOL Upgrade Advisor (new offering; replaces COBOL Migration Assistant)
  - Product page: https://www.ibm.com/products/cobol-upgrade-advisor-zos

Experience more with IBM

Visit us at the IBM Booth #113

After a full day of technical sessions, take a break with us!

Connect with our experts, snap a photo with the z17 Plexi or the latest Telum II, and get an up-close look at our Spyre Accelerator.

Come back each day for fresh topics and demos at our expert stations.

Think 2026

Join 5000+ senior business and technology leaders who are seizing the AI revolution to unlock unprecedented growth and productivity at Think 2026.

Find out more information using the QR code below.

IBM Digital Asset Haven

IBM Digital Asset Haven is the operational backbone for financial institutions and regulated enterprises entering the digital asset economy.

Find out more information using the QR code below.