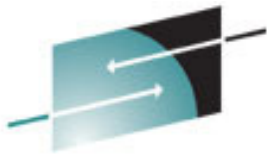
What's New in z/OS Language Environment – C runtime Library?



John Monti

IBM Poughkeepsie

jmonti@us.ibm.com

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- CICS®
- IMS
- Language Environment®
- OS/390®
- z/OS®

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

IEEE is a trademark in the United States and other countries of the Institute of Electrical and Electronics Engineers, Inc.

POSIX® is a registered Trademark of The IEEE.

SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, or service names may be trademarks or service marks of others.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Agenda

- What's New in z/OS V1.12?

- Additional information available in Appendix:
- What's New in z/OS V1.11?

What's New in z/OS V1.12?

- BSAM >64K Track support for type=record
- C++ TR1 support
- realloc() optimization
- VSAM EA KSDS

BSAM >64K Track support for type=record

- Provide support for large format sequential data sets greater than 65,535 tracks/volume opened for **BSAM (seek) under record I/O**.
  - Read, write, reposition
  - Must be opened **type=record** without specifying **noseek**
  - ftell() and fseek() continue to be limited to 2G-1 records.
  - fgetpos() and fsetpos() may reposition past the 2G-1 limit with the following restrictions:
    - **The data set contains 256 TB - 256 bytes or less**
    - **The data set uses 2 GB blocks or less**

C++ TR1 support

- Exposes the C99 namespace and provides C++ overloads for math functions.
- Expose the functionality by either defining _TR1_C99 or __IBMCPP_TR1__
- The focus of LI178 was Chapter 8, ‘C’ compatibility, of the ISO/IEC DTR 19768 Technical Draft Technical Report on C++ Library Extensions Report.

realloc() optimization

- The C/C++ function realloc() supports a new environment variable
  - _CEE_REALLOC_CONTROL
    - Parameter 1 – Lower bound threshold
      - The number of bytes above which the tolerance percentage (parm 2) will be applied
    - Parameter 2 – Tolerance Percentage
      - The percentage of extra storage to be obtained
      - 0 to 100

realloc() optimization

- Example
  - _CEE_REALLOC_CONTROL=100,20
    - First request is for 80 bytes
      - Storage obtained as normal
    - A request to change this storage to 90 bytes
      - Storage obtained as normal
    - A request to change this storage to 100 bytes
      - At or above threshold, percentage is applied
      - Storage obtained is 120 bytes (100 + 100 * 20%)
    - A request to change this storage to 110 bytes
      - No storage need be obtained (we already have 120 bytes)

realloc() optimization

- Can be very useful for programs that make many requests to reallocate storage larger than originally requested.
  - Many string manipulation routines make heavy use of storage reallocation.
- If tolerance percentage is 0 or _CEE_REALLOC_CONTROL is not set no change in behavior.

VSAM EA for KSDS with alternate indexes

- z/OS XL C/C++ applications can now read/write/position beyond the 4GB boundary in VSAM KSDS data sets using alternate index keys.
  - Using fopen() or freopen() against an existing VSAM KSDS extended addressable data set using an alternate index pathname
  - Once opened, using other FILE functions like flocate() to process the stream
- DOC APAR PK83456 extends this support back to z/OS 1.8

The End..

Thank you!

Appendix

- Decimal Floating Point

```
2 (and WithMetalWings) for this hide. Beautiful views further down the trail. Thanks fo
```

Appendix – Decimal Floating Point

- **<math.h>**
  - The following math functions are new for C/C++ decimal floating point:

cbrtd32()	cbrtd64()	cbrtd128()
expm1d32()	expm1d64()	expm1d128()
exp2d32()	exp2d64()	exp2d128()
fmad32()	fmad64()	fmad128()
fmodd32()	fmodd64()	fmodd128()
hypotd32()	hypotd64()	hypotd128()
log1pd32()	log1pd64()	log1pd128()
log2d32()	log2d64()	log2d128()
quantexpd32()	__remquod32()	quantexpd64()
__remquod64()	quantexpd128()	__remquod128()

Appendix – Decimal Floating Point

- **<math.h>**
  - For C++ applications, the following functions are overloaded for the _Decimal32, _Decimal64, and _Decimal128 data types:

cbrt()	expm1()
exp2()	fma()
fmod()	hypot()
log1p()	quantexp()
log2()	

- For example: _Decimal32 cbrt(_Decimal32)

What's new in z/OS R11?

- Decimal Floating Point Part 3 Support
- C/C++ compiler dependencies
- Globalization White Paper 2007
- CCSID functions under CICS
- File I/O Tracing

Decimal Floating Point Support - Part 3

- The new decimal floating point functionality made available in this release is an expansion of the initial support from z/OS V1R9. C/C++ applications can now take advantage of functions and macros that were previously unavailable.
  - Note: APAR PK71899 rolls back the V1R11 decimal floating point support to V1R8.
- **The following support is added for decimal floating point in z/OS V1R11:**
  1. New decimal floating point math functions are provided (see appendix)
  2. New conversion specifiers for printf() family of functions are provided
     - %a and %A are provided for the printf() family of functions to be used with Decimal Floating Point types.
     - These new conversion specifiers determine which type of formatting style is to be used by printf(). Either e/E or f/F style formatting.

Decimal Floating Point Support - Part 3

- Hardware with the Decimal Floating Point Facility must be installed.
- The DFP and ARCH(7) compiler options are required.
- The __STDC_WANT_DEC_FP__ feature test macro must be defined.

C/C++ Compiler Dependencies

- inline versions of four wmem* functions (performance)
- extended character types (capability)
- detect accidental use of Language Environment C headers under Metal C (usability)

C/C++ Compiler Dependencies

- inline versions of four wmem* functions (performance)
  - Using inline wmem* functions, the customer can:
    - Improve application performance by using the inline versions of wmemchr(), wmemcmp(), wmemcpy(), and wmemset()
    - The inline versions of functions always perform better because it eliminates the overhead of a library call
    - The inline versions are not available under LP64

C/C++ Compiler Dependencies

- To use inline wmem* functions:
  - #define appropriate feature test macro
  - #include <wchar.h>
  - use wmemchr(), wmemcmp(), wmemcpy(), or wmemset()
  - compile with ARCH(7)
  - compile with z/OS XL C/C++ V1R11 compiler
  - do not use LP64 compile option

C/C++ Compiler Dependencies

- Using extended character types, the customer can:
  - Use certain width character types, char16_t and char32_t, that are guaranteed consistent across platforms (where the types are supported)
- To use extended character types:
  - #include <uchar.h>
  - use char16_t or char32_t types
  - compile with LANGLVL(EXTENDED) under XL C
  - compile with z/OS XL C/C++ V1R11 compiler

C/C++ Compiler Dependencies

- Using Metal C header detection, the customer can:
  - Detect misuse of Language Environment C headers during compilation of Metal C application
  - This change provides earlier problem detection should any Language Environment C supported feature be attempted to be used in a Metal C application
  - when Language Environment C headers are included in a compile that uses the XL C METAL option, an error message will be issued from each Language Environment C header that is processed

Globalization White Paper 2007

- A new locale for Serbia, sr_RS.UTF-8 is provided
  - This support provides the ability for XL C/C++ applications to now be run under the Serbia locale
  - The new locale for Serbia is an ASCII locale

CCSID functions under CICS

- CCSID functions are now supported under CICS
  - __toCcsid()
  - __CSNameType()
  - __toCSName()
  - __CcsidType()
- Provides the ability for XL C/C++ applications running under CICS to convert between code set names and their CCSID and also determine the type for a given code set name or CCSID (ASCII or EBCDIC)

File I/O Tracing

- Generate a high-level trace for C/C++ file I/O
  - The file I/O trace is controlled with an environment variable
  - By default, file I/O tracing will be turned off
  - The function trace is limited to functions that work with a (FILE *) stream
  - Significantly reduce time spent to recreate error
  - Improve accuracy and focus of problem determination
  - This is not meant for first-failure data capture
  - The initial implementation adds the framework for more detailed levels of trace information

File I/O Tracing

- The File I/O Trace is invoked by:
  - An Environment Variable
    - _EDC_IO_TRACE
    - **Description:**
      - Indicates which files to perform file I/O tracing on, the level of detail to provide for file I/O tracing, and the trace buffer size to use for each file.
      - The _EDC_IO_TRACE format is: _EDC_IO_TRACE=(*Filter*,*Detail*,*Buffer Size*)
  - Details of the values and default settings are described in the **C/C++ Programming Guide**

File I/O Tracing

- Locating the trace file:
  - Under TSO or BATCH
    - DDname EDCTRACE
    - Batch log (SYSOUT=* by default)
  - Under z/OS UNIX System Services
    - Directory found in environment variable *_CEE_DMPTARG*
    - The current working directory, if the directory is writable
    - The directory found in environment variable TMPDIR
    - The /tmp directory

The name of this file uses the following format: /path/EDCTRACE.Date.Time.Pid