#SHAREorg

SHARE
Technology · Connections · Results

Enterprise PL/I 4.2 Highlights

Peter Elderon
elderon@us.ibm.com

March 2012
Session 10756

IBM compilers and you

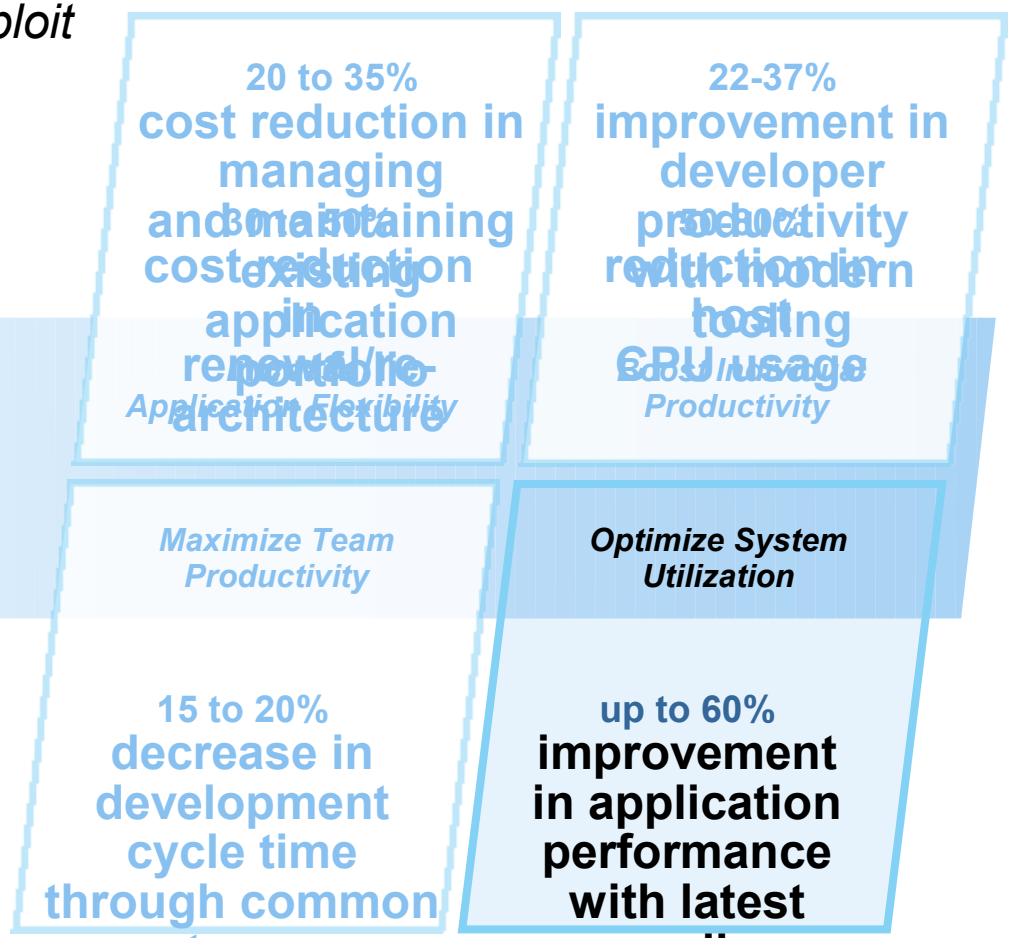
Optimize your infrastructure by upgrading to the latest IBM compilers

Latest compilers and middleware exploit System z for maximum performance

- Enterprise COBOL for z/OS v4.2
- Enterprise PL/I for z/OS v4.2
- z/OS XL C/C++ v1.13

- CICS v4.2
- DB2 for z/OS v10.0
- IMS for z/OS v12.0

Enterprise Modernization



*Sources of these benchmarks are available upon request

With advanced compiler technology

Exploit new advanced optimization technology and new hardware in C/C++, PL/I , Java, and Fortran compilers

***Same* hardware + *NEW* compiler**
= *Increased* Performance

***NEW* hardware + *NEW* compiler**
= *Maximum* Performance

- z/OS XL C/C++ v1.13 on zEnterprise196* delivers up to *60%* performance improvement
- Enterprise PL/I V4.2 on zEnterprise196* delivers up to *10%* performance improvement
- XL C/C++ v11, XL Fortran v13 on POWER7 delivers *industry leading* SPEC® CPU2006 performance*

✓ Lowers capital outlays by increasing capacity and improving performance of application suite

*Sources of these benchmarks are available upon request

With improved compiler middleware support

New releases of COBOL, PL/I and C/C++ provide improved support for middleware via:

- Integrated CICS and SQL translators
  - COBOL, PL/I and C/C++
  - Enterprise PL/I v4.2 improved performance of processing SQL source by up to 40%
- Programming support for new middleware features
  - CICS co-processor options, DB2 features (e.g. multiple-row fetch…)
  - Support for new SQL new data types and SQL syntax first introduced in DB2 v9
- Problem determination support with program listings and Debug Tool
  - Display SQL and CICS options in effect in COBOL and PL/I listing
  - Debug applications written in COBOL, PL/I, C/C++ in CICS, DB2, IMS environments
- XML Support
  - PL/I and COBOL programs can parse and generate XML documents

Follow these best practices

- Upgrade compilers when you upgrade System z hardware, or Middleware (CICS, DB2, IMS)
  - Minimize quality assurance effort
  - Maximize performance
  - Leverage compiler support for new middleware features
  - Improve debugging and programmability
- Recompile only parts that are changed and “hot spots” to improve performance
- Leverage new compiler features to modernize existing business critical applications
  - “Rip and Replace is expensive and risky
  - Modernization promotes reuse and delivery of new solution at lower cost, lower risk, and shorter delivery time
- Use Rational development tools to improve programmer productivity, and help attract new talent
  - Rational Developer for z, Rational Developer for z UT, Rational Team Concert

Enterprise 4.2

- schnell
- lecker
- nach Wunsch

schnell

zEnterprise exploitation

z196

- The new z196 hardware was introduced in 2010
- Enterprise PL/I immediately provided significant exploitation of the new hardware under the ARCH(9) option
- The 4.2 release expands that ARCH(9) support

High-word Facility

- This facility adds a new set of instructions which consider the high-word of the 64-bit GPRs to be self-contained 32-bit registers.
- The compiler now exploits this facility under the HGPR and OPT(3) options
- However, for now, this exploitation is limited to the generation of the BRCTH Branch-Relative-on-Count-High instruction in some loops

High-word Facility

- So, when given this code

```
loop: proc(a)
  options( nodescriptor )
  returns(fixed bin(31) byvalue);

  dcl a(100) fixed bin(31) connected;
  dcl jx  fixed bin(31);
  dcl sum fixed bin(31);
  sum = 0;
  do jx = 1 to hbound(a);
    sum += a(jx);
  end;
  return( sum );
end;
```

High-word Facility

- **Under 4.1 and the options HGPR OPT(3) ARCH(9), the heart of the generated code is**

```
000046   41F0   0000                    LA      r15,0
00004A   4100   0001                    LA      r0,1
00004E   5810   1000                    L       r1,_addrA(,r1,0)
000052   41E0   0004                    LA      r14,4
000056                          @1L2    DS      0H
000056   E3FE   1FFC   FF5E             ALY     r15,_shadow1(r14,r1,-4)
00005C   A70A   0001                    AHI     r0,H'1'
000060   A70E   0064                    CHI     r0,H'100'
000064   41E0   E004                    LA      r14,#AMNESIA(,r14,4)
000068   A7D4   FFF7                    JNH     @1L2
```

High-word Facility

- **While under 4.2, the heart of the generated code is 6 bytes smaller**

```
000046   41F0   0000                    LA       r15,0
00004A   5810   1000                    L        r1,_addrA(,r1,0)
00004E   41E0   0004                    LA       r14,4
000052   C008   0000   0064             IIHF     r0,F'100'
000058                           @1L2   DS       0H
000058   E3FE   1FFC   FF5E             ALY   r15,_shadow1(r14,r1,-4)
00005E   41E0   E004                    LA    r14,#AMNESIA(,r14,4)
000062   CC06   FFFF   FFFB             BRCTH    r0,@1L2
```

Population-count Facility

- The new POPCNT instruction provides a count of the number of one bits in each of the eight bytes of the input GPR.
- Each byte in the output GPR contains an 8-bit binary integer in the range of 0-8 holding the count for the corresponding byte.
- PL/I now exploits this facility via the new POPCNT built-in function

Population-count Facility

- On z/OS, this built-in function requires an ARCH level of 9 or higher

  POPCNT(‘01020304’xn) returns ‘01010201’xn
  POPCNT(‘05060708’xn) returns ‘02020301’xn

- And, if x has the attributes FIXED BIN(31), then

  ISRL(POPCNT(x) * ’01010101’xn, 24)

- returns the number of bits in x equal to 1

Extended-float facility

- The compiler now exploits this facility to inline conversions between IEEE float and FIXED BIN(63). In particular, it will generate

- The following new BFP instructions :
  - CONVERT FROM LOGICAL (CXLFBR, CDLFBR, CELFBR, CXLGBR, CDLGBR, CELGBR)
  - CONVERT TO LOGICAL (CLFXBR, CLFDBR, CLFEBR, CLGXBR, CLGDBR, CLGEBR)

- The following new DFP instructions :
  - CONVERT FROM FIXED (CXFTR,CDFTR)
  - CONVERT FROM LOGICAL (CXLGTR,CDLGTR, CXLFTR, CDLFTR)
  - CONVERT TO FIXED (CFXTR, CFDTR)
  - CONVERT TO LOGICAL (CLGXTR,CLGDTR, CLFXTR, CLFDTR)

Extended-float facility

- So, when given this code

```
*process
  arch(9) float(dfp) limits(fixedbin(31,63)) opt(3);

  cfdtr:
    proc(d)
    returns( fixed bin(31) byvalue );

    dcl d              float dec(16);
    dcl n              fixed bin(31);

    n = d;
    return( n );
  end;
```

Extended-float facility

- **Under 4.1, the heart of the generated code is the scary**

```
000046   58E0   1000                    L       r14,_addrD(,r1,0)
00004A   C010   0000   002F             LARL    r1,F'47'
000050   6800   E000                    LD      f0,_shadow1(,r14,0)
000054   B3E1   9000                    CGDTR   r0,b'1001',f0
000058   18F0                           LR      r15,r0
00005A   EB00   0020   000C             SRLG    r0,r0,32
000060   5900   1000                    C       r0,+CONSTANT_AREA(,r1,0)
000064   A774   0005                    JNE     @1L4
000068   C2FF   8000   0000             CLFI    r15,F'-2147483648'
00006E                          @1L4    DS      0H
00006E   A724   0007                    JH      @1L3
000072   A708   FFFF                    LHI     r0,H'-1'
000076   C0F9   8000   0000             IILF    r15,F'-2147483648'
00007C                          @1L3    DS      0H
00007C   1200                           LTR     r0,r0
00007E   A774   0005                    JNE     @1L6
000082   C2FF   7FFF   FFFF             CLFI    r15,F'2147483647'
000088                          @1L6    DS      0H
000088   A744   0005                    JL      @1L2
00008C   C0F9   7FFF   FFFF             IILF    r15,F'2147483647'
```

Extended-float facility

- But under 4.2, the heart of the generated code is the simple

```
000046   5810   1000          L       r1,_addrD(,r1,0)
00004A   6800   1000          LD      f0,_shadow1(,r1,0)
00004E   B941   90F0          CFDTR   r15,b'1001',f0
```

Additional performance enhancements

Loop unrolling

- The new UNROLL compiler option controls whether loops are unrolled
- The default is UNROLL(AUTO) which lets the compiler determine when loops are unrolled – and which matches what the previous releases did
- UNROLL(NO) suppresses all loop unrolling
- UNROLL(AUTO) may produce bigger, but faster object code

Loop unrolling

- So, when given this code

```
unroll: proc(a)
  options( nodescriptor )
  returns(fixed bin(31) byvalue);

  dcl a(10) fixed bin(31) connected;
  dcl jx  fixed bin(31);
  dcl sum fixed bin(31);
  sum = 0;
  do jx = 1 to 10;
    sum += a(jx);
  end;
  return( sum );
end;
```

Loop unrolling

- **Under UNROLL(NO), the heart of the generated code is the short**

```
000046   5810   1000              L       r1,_addrA(,r1,0)
00004A   41F0   0000              LA      r15,0
00004E   41E0   0004              LA      r14,4
000052   4100   000A              LA      r0,10
000056   A71A   FFFC              AHI     r1,H'-4'
00005A                    @1L2    DS      0H
00005A   5EFE   1000              AL      r15,_shadow1(r14,r1,0)
00005E   41E0   E004              LA      r14,#AMNESIA(,r14,4)
000062   A706   FFFC              BRCT    r0,@1L2
```

Loop unrolling

- **While under UNROLL(AUTO), it is the longer but faster**

```
000046   5810   1000          L       r1,_addrA(,r1,0)
00004A   58F0   1000          L       r15,_shadow1(,r1,0)
00004E   5EF0   1004          AL      r15,_shadow1(,r1,4)
000052   5EF0   1008          AL      r15,_shadow1(,r1,8)
000056   5EF0   100C          AL      r15,_shadow1(,r1,12)
00005A   5EF0   1010          AL      r15,_shadow1(,r1,16)
00005E   5EF0   1014          AL      r15,_shadow1(,r1,20)
000062   5EF0   1018          AL      r15,_shadow1(,r1,24)
000066   5EF0   101C          AL      r15,_shadow1(,r1,28)
00006A   5EF0   1020          AL      r15,_shadow1(,r1,32)
00006E   5EF0   1024          AL      r15,_shadow1(,r1,36)
```

REFER

- Code that uses elements of structures with multiple REFERs used to be very expensive: each reference used a costly library call to remap the structure
- As of 4.1, for structures where all the elements are byte-aligned, those calls are avoided and straightforward inline code generated
- If all elements are byte-aligned, no padding is possible and thus the address calculations are relatively simple
- To insure all elements are byte-aligned
  - Specify UNALIGNED on the level-1 part of the declare
  - Declare any NONVARYING BIT as ALIGNED

REFER

- But with 4.1, if the STG built-in function was applied to such a structure, library calls would still be made
- As of 4.2, no library calls will be made for STG when applied to such structures
- This can be very useful if you want to use functions like PLIMOVE to copy an entire structure using multiple REFERs
- And this is needed by some IMS tools

UTF string handling

- When ULENGTH and USUBSTR are applied to CHARACTER strings, the compiler will now generate inline code (rather than call a library routine)
- This makes these functions much, much faster
- However, it also means that ERROR will not be raised if the source is invalid UTF-8
- The UVALID function can still be used to test for validity

lecker

Menu

- Vorspeise
  - Errol Morris and the iron triangle
- Hauptgang
  - New and improved SQL preprocessor
- Nachtisch
  - Improved multi-row fetch support - and more

Vorspeise

The iron triangle

- There is also an old engineering saying that you cannot make a product
  - fast, cheap, and reliable
- You can have any 2, but not all 3
  - BMW X6 M — fast and reliable, but not cheap
  - VW Beetle — cheap and reliable, but not fast
  - Used Ford Mustang — fast and cheap, but not reliable
- This rule has also been applied to project management where the rule is that you can have only two of: fast, cheap, and good

“Fast, Cheap & Out of Control”

- In 1997, Errol Morris released “Fast, Cheap & Out of Control” about
  - a lion trainer
  - a topiary (Formschnitt) sculptor
  - a hairless-mole-rat specialist
  - an MIT robot scientist
- The MIT scientist designs bug-like robots and had written a technical paper, “A Robot Invasion of Space”, advocating the use of many fast and small, if not reliable, robots as the way to explore space
- No SQL was used

Hauptgang

Fast, small, and powerful

- The 4.2 release contains a completely redesigned SQL preprocessor that is
  - Fast – 20-50% faster than the 4.1 preprocessor
  - Small - more than 8 times smaller than the 4.1 module
  - Powerful – many restrictions removed
- It should also be more reliable as it is built on the same code base as the macro and CICS preprocessors - which have had far fewer APARs

SQL preprocessor improvements

- Name scoping always in effect and with no restrictions
- The SCOPE option in 3.9 and 4.1 had these restrictions
  - The data lists in GET and PUT statements must not include data-list items with Type 3 DO specifications.
  - The following keywords must not be used as variable names: BEGIN, DO, END, PACKAGE, PROC, PROCEDURE and SELECT
- 3.9 and 4.1 had NOSCOPE as the default partly because of these restrictions
- 4.2 has dropped the (NO)SCOPE option, and name scoping is always in effect and without any restrictions

SQL preprocessor improvements

- **SQL TYPE supported as a first-class PL/I attribute**
- **Anywhere you could use a PL/I attribute such as FIXED BIN or CHAR, you can now use any of the SQL TYPE attributes, e.g.**
  - **dcl blobs(10) *sql type is blob(100k)* based;**
- **But unlike “FIXED BIN”, no keywords may break up “SQL TYPE IS …”**

SQL preprocessor improvements

- Declare statements are fully and correctly processed, including
- These attributes are now honored and may be used in host variables: PRECISION and DIMENSION
  - The old preprocessor objected to DCL A FIXED BIN PREC(31) and to DCL B DIM(3) FIXED BIN(15)
- These attributes are now recognized and would cause any host variable with them to be unusable: UNSIGNED and COMPLEX
  - The old preprocessor accepted DCL C FIXED BIN(16) UNSIGNED and incorrectly viewed it as a 4 byte integer

SQL preprocessor improvements

- DEFAULT compiler suboptions are now honored
- DEFAULT(ANS/IBM) is honored when completing numeric attributes
- Also honored are the suboptions:
  - (NON)NATIVE
  - ASCII / EBCDIC
  - (NO)EVENDEC
  - SHORT(HEX / IEEE)
- Plus the RULES((NO)LAXCTL) option is also honored

SQL preprocessor improvements

- **PACKAGEs fully supported**
- **For a PACKAGE, declares inserted by the SQL preprocessor are now put into each outermost PROC as needed**
- **The old preprocessor put them at the PACKAGE level - which made the code non-reentrant unless it was compiled with the RENT option**

SQL preprocessor improvements

- **TWOPASS option effectively always on**
- **As in normal PL/I code, declarations and statements can appear in any order**
- **And there is no extra processing cost for this (as there is also none in the compiler itself)**

SQL preprocessor difference: LOB representation

- SQL TYPE has a different representation for large objects
- The LOB(DB2 | PLI) option used to control this – but that option is gone
- If you have code that “knows” the representation of SQL TYPE LOB data, that code will almost certainly have to be changed
- But, since SQL TYPE is now essentially a full-fledged attribute, now your code doesn’t need to “know” this
- For example:

SQL preprocessor difference: LOB representation

- The structure element XML_DOC_STRUC knows the representation of a CLOB

```
DCL
   1 DOCM_STRUC,
     2 DOC_ID                    FIXED BIN(31),
     2 DOC_TYPE                  CHAR(1),
     2 XML_DOC_STRUC,
       3 XML_DOC_ARRY_LENGTH     FIXED BIN(31),
       3 XML_DOC_ARRY_DATA,
         4 XML_DOC_DATA1(3)      CHAR(32767),
         4 XML_DOC_DATA2         CHAR(4099);
DCL ID_ARRAY(5)                FIXED BIN(31);
DCL TYPE_ARRAY(5)              CHAR(1);
DCL XML_DOC_ARRAY(5)           SQL TYPE IS XML AS CLOB(100K);

EXEC SQL FETCH NEXT ROWSET FROM DOCM_CSR FOR 5 ROWS
           INTO  :ID_ARRAY
                ,:TYPE_ARRAY
                ,:XML_DOC_ARRAY;

XML_DOC_STRUC = XML_DOC_ARRAY(I);
```

SQL preprocessor difference: LOB representation

- With 4.2, it can – and must – be declared as SQL TYPE …

```
DCL
   1 DOCM_STRUC,
     2 DOC_ID                FIXED BIN(31),
     2 DOC_TYPE              CHAR(1),
     2 XML_DOC_STRUC         SQL TYPE IS XML AS CLOB(100K);

DCL ID_ARRAY(5)              FIXED BIN(31);
DCL TYPE_ARRAY(5)            CHAR(1);
DCL XML_DOC_ARRAY(5)         SQL TYPE IS XML AS CLOB(100K);
```

SQL preprocessor difference: LOB representation

- If you have code that has this dependency, it will be easy to spot
- The compiler will issue a severe message for the assignment statement
- And if you don't get any such message, then your code is ok as is

SQL preprocessor difference: messages

- SQL preprocessor messages are now in the same series as the messages produced by the MACRO and CICS preprocessors (since they all share some common code)
- So, the message numbers and text for SQL messages have all changed
- So, if you are using the EXIT compiler option to suppress or to change the severity of any SQL messages, you will have to change the message table

SQL preprocessor difference: messages

- Backend SQL messages also now have an IBM “facility id” (as do all other messages including those produced by the CICS backend)
- So, if you are using the EXIT compiler option to suppress or to change the severity of any SQL backend messages, you will have to use change the EXIT itself
- There is a full example of how to do this in the 4.2 Programming Guide

Nachtisch

BY DIMACROSS

- The new BY DIMACROSS assignment statement makes it possible to write nicer code after performing multi-row fetches
- For example, given

```
dcl 1 A, 2 B fixed bin, 2 C fixed dec, 2 D char(10);
dcl 1 XA(100) dimacross like A;
```

- After a multi-row fetch, you might want to assign the JX element of XA to A

```
A = XA( JX );
```

- But this is illegal since XA is not an array – only its elements are!

BY DIMACROSS

- But 4.2 supports a new statement to help with this; given

```
dcl 1 A, 2 B fixed bin, 2 C fixed dec, 2 D float bin;
dcl 1 XA(100) dimacross like A;
```

- You can now write

```
A = XA, BY DIMACROSS( JX );
```

- Which will assign the JXth element in each of the arrays in XA to the corresponding element in A

BY DIMACROSS

- You could use this to swap two rows; given

```
dcl 1 A, 2 B fixed bin, 2 C fixed dec, 2 D float bin;
dcl 1 T1 like A, 1 T2 like A;
dcl 1 XA(100) dimacross like A;
```

- You could swap row 1 and 17 via

```
T1 = XA, BY DIMACROSS( 1 );
T2 = XA, BY DIMACROSS( 17 );
XA = T2, BY DIMACROSS( 1 );
XA = T1, BY DIMACROSS( 17 );
```

BY DIMACROSS

- You could also use this to sum across the rows; given

```
dcl 1 A, 2 B fixed bin, 2 C fixed dec, 2 D float bin;
dcl 1 XA(100) dimacross like A;
```

- You could sum across the rows via

```
A = 0;
DO JX = 1 TO 100;
   A = A + XA, BY DIMACROSS( JX );
END;
```

HBOUNDACROSS

- The new HBOUNDACROSS and LBOUNDACROSS built-in functions make it possible to write even nicer code for this; namely, given

```
dcl 1 A, 2 B fixed bin, 2 C fixed dec, 2 D float bin;
dcl 1 XA(100) dimacross like A;
```

- You could sum across the rows via

```
A = 0;
DO JX = LBOUNDACROSS(XA) TO HBOUNDACROSS(XA);
   A = A + XA, BY DIMACROSS( JX );
END;
```

INDICATORS

- If you use an indicator array with a structure in an SQL statement, the array should have as many elements as the structure
- This means that if the structure changes, you also have to change the declare for the indicator array
- The new INDICATORS built-in function eliminates this task
- It counts the number of elements in a structure and can be used to declare an array with that as it upper bound

INDICATORS

- So, given a declare such as:

```
dcl
  1 a,
    2 b   fixed bin,
    . . .
    2 x   fixed bin;
```

You can then declare an indicator array for use with this structure as:

```
dcl inx( indicators(a) ) fixed bin(15);
```

<> as not-equals

- SQL syntax uses <> to mean “not equals”
- This is now supported by all the preprocessors and by the compiler itself
- And <> has an advantage over ^=
  - it does not depend on a code point that varies across code pages

Nach Wunsch

Enhanced XML generation

XMLOMIT

- The new XMLOMIT attribute lets you direct XMLCHAR to suppress some fields
- in particular, it can be used to suppress
  - A string field that compares equal to the null string (‘’)
  - A numeric field that compares equal to zero
- But it is not permitted on structures, unions or fields named with an *
- So there is no suppression of structures

XMLOMIT

- So, when given

```
dcl
   1 order,
     2 orderNr              char(20) init('1729'),
     2 customer,
        3 name              char(32) init(‘euler'),
        3 firstName         char(24) init(‘leonhard'),
        3 special           char(35) init('');
```

XMLOMIT

- **XMLCHAR would generate**

```
<order>
  <orderNr>1729</orderNr>
  <customer>
    <name>euler</name>
    <firstName>leonhard</firstName>
    <special></special>
  </customer>
</order>
```

XMLOMIT

- **But with XMLOMIT added as in**

```
dcl
   1 order,
     2 orderNr              char(20) init('1729'),
     2 customer,
        3 name              char(32) init(‘euler'),
        3 firstName         char(24) init(‘leonhard'),
        3 special           xmlomit char(35) init('');
```

XMLOMIT

- **XMLCHAR would generate the simpler**

```
<order>
  <orderNr>1729</orderNr>
  <customer>
    <name>euler</name>
    <firstName>leonhard</firstName>
  </customer>
</order>
```

XMLATTR

- The new XMLATTR attribute lets you direct that XMLCHAR emit specified fields as attributes
- It cannot be used on arrays, structures, unions, or fields named with an *
- It also cannot be used on an element of a structure if previous elements with the same parent do not have the XMLATTR attribute
- So, there is no regrouping

XMLATTR

- So, with XMLATTR added as in

```
dcl
   1 order,
     2 orderNr              char(20) init('1729'),
     2 customer,
        3 name              xmlattr char(32) init(‘euler'),
        3 firstName         xmlattr char(24) init(‘leonhard'),
        3 special           xmlomit char(35) init('');
```

XMLATTR

- **XMLCHAR would generate XML with these fields as attributes**

```
<order>
  <orderNr>1729</orderNr>
  <customer name='euler' firstName='leonhard'>
  </customer>
</order>
```

New (sub)options for better quality

PPLIST (GRZ - MR0322112751)

- The new PPLIST option controls whether the compiler keeps or erases the part of the listing generated by each preprocessor phase.
- PPLIST(KEEP) is the default
- Under PPLIST(ERASE), the compiler will “erase” any preprocessor listing if that preprocessor produced no messages
- So specifying PPLIST(ERASE) INSOURCE FLAG(W) would produce a listing that was small except when a warning message was generated

NOLAXENTRY (Racon – MR1001105028)

- Under RULES(NOLAXENTRY), the compiler will flag all unprototyped ENTRY declares, d.h. all ENTRY declares that do not specify a parameter list
- With 4.2, NOLAXENTRY can be qualified as STRICT or LOOSE
- RULES(NOLAXENTRY(STRICT)) is the default (for compatibility) and will cause the compiler to flag all unprototyped ENTRY declares
- But under RULES(NOLAXENTRY(LOOSE)), the compiler will flag unprototyped ENTRY declares only if they are not OPTIONS(ASM)
- The LOOSE suboption is probably much more useful

NOSELFASSIGN (Telcordia – MR1222106055)

- **Under RULES(NOSELFASSIGN), the compiler will flag all assignments where the source and the target are the same**

- **This will make it easier to catch Fingerfehler**

NOLAXRETURN (GRZ - MR0216116237)

- Under RULES(NOLAXRETURN), the compiler will generate code to raise ERROR
  - when a RETURN with an expression is executed inside a procedure coded without RETURNS
    - or
  - when a RETURN without an expression is executed inside a procedure coded with RETURNS
- Note that this is one of very few RULES suboptions that change the code generated

DSN and RULES (BayernLB)

- The compiler will no longer flag names starting with DSN under
  - RULES(NOLAXENTRY)
- The compiler will no longer flag names starting with DSN or SQL under
  - RULES(NOUNREF)
- This is reasonable since these names are usually not ones the programmer has declared

CHECK(STORAGE) (CreditSuisse)

- The STORAGE suboption of the CHECK option has been restored
- It was dropped in 4.1 – CHECK(NOSTORAGE) was always in effect
- But since code compiled with CHECK(NOSTORAGE) cannot be mixed with code compiled with CHECK(STORAGE), this meant that 4.1 would require a complete recompile
- It was also restored to 4.1 via PTF UK68593

Miscellaneous user requirements

Restrictions lifted

- Typed structures are now supported in DebugTool
  - The code must be compiled with the 4.2 compiler
  - Runtime PTF for APAR PM30489 must be applied
- Comparisons of POINTER to ‘’ and ‘’b are now allowed
  - The semantics match those in the assignment of ‘’ to a POINTER
  - The is requirement MR0302115849 from GRZ

%INCLUDE enhancements

- The maximum number of distinct include files allowed in a single compilation has been increased from 2047 to 4095
  - This is requirement MR0517112942 from Huk
  - This was also added to 4.1 via PTF UK67957
- The source for %INCLUDE statements will now be bracketed with a BEGIN comment and an END comment
  - Only when the MACRO preprocessor does the INCLUDE
  - This is requirement MR0718114811 from Axa

Enhanced INONLY (Racon - MR0828104258)

- The 3.9 release introduced the INONLY, INOUT and OUTONLY
- However, apart from serving to make code self-documenting, the only effects they had were
  - The compiler would not flag dummy arguments for INONLY parameters
  - The compiler would flag (possibly) uninitialized OUTONLY arguments
- But now, the compiler will apply the NONASSIGNABLE attribute to any INONLY parameter (and hence flag any assignments to it)
- Furthermore, the compiler will flag any structure declared as INONLY but containing an element with the ASSIGNABLE attribute

BASED with nonconstant extents

- With OS PL/I, this was possible only via REFER
- The documentation for Enterprise PL/I matched that, but the compiler allowed nonconstant extents anyway – and users started using it
- We started flagging this with 3.9, but quickly changed back to allowing it
- The documentation has now also been updated to say it is ok

In review

schnell

- zEnterprise exploitation
- Performance enhancements
  - Control of loop unrolling
  - Inlining STG of structures using REFER
  - Inlining of ULENGTH and USUBSTR

lecker

- Improved SQL support
  - Faster, smaller and more powerful preprocessor
  - BY DIMACROSS assignments
  - HBOUNDACROSS built-in function
  - INDICATORS built-in function

nach Wunsch

- Enhanced XML generation
  - XML attributes
  - Suppression of null fields
- Requested new (sub)options for better quality
  - PPLIST option
  - New RULES (sub)options
- Miscellaneous user requirements
  - Restrictions lifted
  - %INCLUDE enhancements

And remember

- COBOL, PL/I, and C/C++ are strategic for System z
  - Committed to continue to deliver value to customers
  - Strong investment commitment from IBM
- IBM compilers are designed to exploit z/Architecture and Middleware (CICS, DB2, IMS)
- IBM compilers provide capability to…
  - Modernize business critical COBOL and PL/I applications with improved Java and XML interoperability
  - Consolidate applications to System z by supporting industry language standards and extensions
  - Reduce programming complexity
    - Advance optimization technology
    - Enhanced problem determination capabilities
    - System program development capability of "Metal C" option
- Rational is committed to delivering leading-edge compilers and application development tools technology to…
  - Maximize application performance and increase system capacity
  - Improve programmer productivity and shorten development cycle
  - Lower TCO and increase return on IT investment

For more information

- System z compilers product pages
  - Enterprise COBOL for z/OS http://www-01.ibm.com/software/awdtools/cobol/zos/
  - Enterprise PL/I for z/OS http://www-01.ibm.com/software/awdtools/pli/plizos
  - z/OS XL C/C++ http://www-01.ibm.com/software/awdtools/czos/

- Documentation
  - Enterprise COBOL for z/OS http://www-306.ibm.com/software/awdtools/cobol/zos/library/?S_CMP=rnav
  - Enterprise PL/I for z/OS http://www-306.ibm.com/software/awdtools/pli/plizos/library/?S_CMP=rnav
  - z/OS XL C/C++ http://www-306.ibm.com/software/awdtools/czos/library/?S_CMP=rnav

- Rational Café https://www.ibm.com/developerworks/rational/community/cafe/
  - Compilers and Application Tools user communities
- Rational Enterprise Modernization sandbox for System z
  - http://www.ibm.com/developerworks/downloads/emsandbox_systemz/index.html
  - Try out Rational Enterprise Modernization tools in a cloud environment

- Main compiler product pages:
  - COBOL http://www.ibm.com/software/awdtools/cobol
  - PL/I http://www.ibm.com/software/awdtools/pli
  - C/C++ http://www.ibm.com/software/awdtools/ccompilers

Learn more at:

- IBM Rational software
- IBM Rational Software Delivery Platform
- Process and portfolio management
- Change and release management
- Quality management
- Architecture management
- Rational trial downloads
- developerWorks Rational
- IBM Rational TV
- IBM Rational Business Partners