IBM Software Group

Enterprise PL/I 4.4 Highlights

March 10, 2014

Rational. software

Go to IBM

Peter Elderon

elderon@us.ibm.com

Enterprise 4.4

- performance
- middleware support
- usability

performance

Listing generation

- **The code in the backend to generate the pseudo-assembler listing has been significantly improved**
- **It is much faster and will use much less cpu**
- **This change was also included in 4.3 in a PTF**

More DFP exploitation in PICTURE conversions

- **When converting PICTURE to FIXED BIN and the source precision was 9 or less, then DFP was used as intermediary (rather than FIXED DEC)**
- **This provided a performance boost in 4.3**
- **It has now been extended to when the source precision is 18 or less**

Example: Picture to Fixed Bin(63)

- **So, for example, when given this code to convert PICTURE to FIXED BIN**

```
pic2int: proc( ein, aus ) options(nodescriptor);

  dcl ein(0:100_000) pic'(15)9' connected;
  dcl aus(0:hbound(ein)) fixed bin(63) connected;
  dcl jx  fixed bin(31);

  do jx = lbound(ein) to hbound(ein);
    aus(jx) = ein(jx);
  end;
end;
```

Example: Picture to Fixed Bin(63)

- **Under 4.3 or ARCH(9), the heart of the loop consists of these 8 instructions**

```
F27E  D0A8  4000    PACK    #pd580_1(8,r13,168),_shadow2(15,r4,0)
D207  D0B0  D0A8    MVC     #pd581_1(8,r13,176),#pd580_1(r13,168)
D100  D0B7  200C    MVN     #pd581_1(1,r13,183),+CONSTANT_AREA(r2,12)
F8F7  D0B8  D0B0    ZAP     #pd582_1(16,r13,184),#pd581_1(8,r13,176)
D20F  D098  D0B8    MVC     _temp1(16,r13,152),#pd582_1(r13,184)
E300  D098  000E    CVBG    r0,_temp1(,r13,152)
1820                LR      r2,r0
EB00  0020  000C    SRLG    r0,r0,32
```

Example: Picture to Fixed Bin(63)

- **While under 4.4 and ARCH(10), it consists of 6 instructions and uses DFP in several of them – but since only the new CXZT instruction refers to storage, the loop runs almost twice as fast**

```
ED0E  1000  00AB    CXZT    f0,#AddressShadow(15,r1,0),b'0000'
B914  0000          LGFR    r0,r0
B3FE  0000          IEXTR   f0,f0,r0
B3E9  9010          CGXTR   r1,b'1001',f0
1841                LR      r4,r1
EB11  0020  000C    SRLG    r1,r1,32
```

Decimal-Floating-Point Zoned-Conversion Facility

- **To summarize some of the lessons from this and the last release:**
  - **A longer set of instructions may be faster than a shorter set**
  - **Reducing storage references helps performance**
  - **Eliminating packed decimal instructions can help performance**
  - **Using decimal-floating-point may improve your code's performance even in program's that have no floating-point data**
  - **The 4.3 PL/I compiler knows when these new ARCH(10) instructions will help and will exploit them appropriately for you**

Inlining of FIXED to WIDECHAR

- **For “nice” FIXED, conversion to WIDECHAR will be inlined by converting to ASCII and then converting the ASCII to WIDECHAR – in 4.3 a library call was made, so this code is much faster**
  - **“nice” FIXED BIN is (UN)SIGNED REAL FIXED BIN(p,0)**
  - **“nice” FIXED DEC is REAL FIXED DEC(p,q) with 0 <= q and q <= p**
- **For FIXED BIN to WIDECHAR, the conversion is done via**
  - **CVD**
  - **EDMK (using special characters in the edit pattern)**
  - **NC (to complete the conversion to ASCII)**
  - **ASCII to UTF-16**

Other UTF-8 and UTF-16 improvements

- **When possible, MVC, rather than MVCLU, is now used in assignments of WIDECHAR to WIDECHAR**
- **The following built-in functions are now collapsed, when their arguments are restricted expressions, at compile-time**
  - **Ulength, Ulength8, Ulength16**
  - **Usubstr**
  - **Usupplementary**
  - **Upos, Uwidth**
- **So, these functions may all be used in restricted expressions themselves**

Middleware improvements - SQL

SQL preprocessor error handling

- **Starting with the 4.2 release, the SQL preprocessor always keep track of the PL/I block structure of the program (so that name resolution is correct)**
- **This is a plus for you – as long as your code is syntactically valid**
- **But for some invalid programs, the error messages produced by the SQL preprocessor were not very helpful**
- **This situation has been vastly improved with the 4.4 release**
- **This is RFE 31207 from IBM Japan (and requested by other customers, too)**

SQL preprocessor error handling

- **So, instead of this 4.2 listing and its one unhelpful message**

```
      1.0       SQLTEST: PROC( XML_PTR;)
      2.0         EXEC SQL INCLUDE SQLCA;
      3.0         BEGIN OPTIONS(INLINE;)
      4.0         END;
      5.0         IF NOT SW1 ! SW2 THEN DO;
      6.0         END;
      7.0         PUT SKIP LIST( 'test';)
      8.0         EXEC SQL COMMIT;
      9.0       END;

IBM3332I W        9.0     The END statement has no matching BEGIN, DO,
                          PACKAGE, PROC, or SELECT. This may indicate a
                          problem with the syntax of a previous statement.
```

SQL preprocessor error handling

- **Under 4.4, you would get these much more helpful messages**

```
IBM3985I S      1.0   Semicolon found before required closing right
                      parenthesis.
IBM3987I S      1.0   Statement must start with a keyword or assignment
                      target.
IBM3985I S      3.0   Semicolon found before required closing right
                      parenthesis.
IBM3987I S      3.0   Statement must start with a keyword or assignment
                      target.
IBM3986I S      5.0   IF statement syntax is invalid.
IBM3985I S      7.0   Semicolon found before required closing right
                      parenthesis.
IBM3987I S      7.0   Statement must start with a keyword or assignment
                      target.
IBM3332I W      9.0   The END statement has no matching BEGIN, DO,
                      PACKAGE, PROC, or SELECT. This may indicate a
                      problem with the syntax of a previous statement.
```

SQL EMPTYDBRM option

- **With 4.2, the SQL preprocessor invokes the DB2 backend only if the code contains at least one EXEC SQL statement**
- **This makes it much faster when the code contains no EXEC SQL**
- **But it also means that no DBRM will be created if there is no EXEC SQL – rather than create an empty DBRM**
- **Some users want an empty DBRM when there is no EXEC SQL, and the new SQL preprocessor option EMPTYDBRM will insure this**
- **The default option is NOEMPTYDBRM**

SQL structures with arrays

- **The 4.2 and 4.3 SQL preprocessor produced an S-level message if a structure containing an array was used as a host variable**
- **It will now accept this usage as long as the host variable is not followed by an indicator variable**
- **This change is available in 4.2 and 4.3 via APAR PM95407**

Commented SQL statements

- **The 4.2 and 4.3 SQL preprocessor includes the original SQL statement in a comment and prevents any contained comments from causing problems by changing any / to \ (a hex E0)**
- **But this included / characters that were in strings in the SQL statement**
- **This would make the listing appear incorrect**
- **Now it changes only */ and then to *>**
- **This change is available in 4.2 and 4.3 via APAR PM95593**

Middleware improvements - IMS

Introduction, part 1

- **The PL/I IMS importer makes use of base 64 encodings to transport some data and also does a lot of work with XML**
- **Each instance contained copies of (sometimes large) functions to help with these tasks, but that limited the number of importers that could be active**
- **So, several new functions have been added to PL/I**
- **These functions are also generally useful outside of IMS**

Base 64

- **Base 64 is used to encode binary data as text**
- **The first 62 “digits” are**
  - **ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789**
- **The final 2 vary according to the encoding scheme. A common choice and the one used by PL/I is to use the “digits”**
  - **+/**
- **All of these 64 characters are code page invariant and from the pure ASCII code page – so they can survive transport unchanged**

Base 64

- **The encoding of data to base 64 and decoding from base 64 back is non-trivial**
- **So, with 4.4, PL/I has made available 2 sets of 2 built-in functions**
  - **base64encode8(p, m, q, n)**
  - **base64decode8(p, m, q, n)**
  - **base64encode16(p, m, q, n)**
  - **base64decode16(p, m, q, n)**
- **This is RFE 31216 from Credit-Suisse**

Base 64

- **Base64encode8 will write UTF-8 (ASCII, in fact) to the target buffer**
- **Base64decode8 requires that the source buffer came from base64encode8**
- **Base64encode16 will write UTF-16 to the target buffer**
- **Base64decode16 requires that the source buffer came from base64encode16**

Base 64

- **All the functions have 4 similar parameters, e.g. base64encode8(p, m, q, n)**
  - **p specifies the address of the target buffer**
  - **m specifies the size in bytes of the target buffer**
  - **q specifies the address of the source buffer**
  - **n specifies the size in bytes of the source buffer**
- **If the address of the target buffer is zero, the number of bytes that would be written is returned.**
- **If the target buffer is not large enough, a value of -1 is returned.**
- **If the target buffer is large enough, the number of bytes written to the buffer is returned**

Base 64

- **Note that If the source length in bits is not a multiple of 6, the result concludes with one or two = symbols**
- **So under base64encode8,**
  - **The 6 byte source ‘please’E will produce the 8 byte UTF8(‘l5OFgaKF’)**

Base 64

- **Note that If the source length in bits is not a multiple of 6, the result concludes with one or two = symbols**

- **So, for example, base64encode8 would produce these results for these 3 strings**

	Source length	Source value	Result length	Result value
	6	‘please’E	8	UTF8(‘l5OFgaKF’)
	5	‘pleas’E	8	UTF8(‘l5OFgaI=’)
	4	‘plea’E	8	UTF8(‘l5OFgQ==’)

XML normalization

- **Two new built-in functions help “normalize” a buffer containing XML:**
- **WHITESPACEREPLACE replaces any of \t, \f, \v, \n, and \r by a UTF blank**
- **WHITESPACECOLLAPSE does that plus it**
  - **Trims all leading and trailing blanks**
  - **Reduces any other sequences of blanks to a single blank**
- **The source buffer must contain UTF-16, not UTF-8**
- **This is RFE 31214 from Credit-Suisse**

XML normalization

- **The functions have the usual 4 parameters:**
  - **p specifies the address of the target buffer**
  - **m specifies the size in bytes of the target buffer**
  - **q specifies the address of the source buffer**
  - **n specifies the size in bytes of the source buffer**
- **If the address of the target buffer is zero, the number of bytes that would be written is returned.**
- **If the target buffer is not large enough, a value of -1 is returned.**
- **If the target buffer is large enough, the number of bytes written to the buffer is returned**

XML cleaning

- **The new XMLCLEAN built-in functions “cleans” a buffer containing XML by**
- **Replacing any invalid UTF by a UTF blank**
- **Replacing any carriage-returns by
**
- **Replacing any of the characters for quote, apostrophe, ampersand, less-than, and greater-than by their predefined entity references**
  - **e.g. & is replaced by &**
- **The source buffer must contain UTF-16, not UTF-8**
- **This is RFE 31214 from Credit-Suisse**

XML cleaning

- **The function has the usual 4 parameters:**
  - **p specifies the address of the target buffer**
  - **m specifies the size in bytes of the target buffer**
  - **q specifies the address of the source buffer**
  - **n specifies the size in bytes of the source buffer**
- **If the address of the target buffer is zero, the number of bytes that would be written is returned.**
- **If the target buffer is not large enough, a value of -1 is returned.**
- **If the target buffer is large enough, the number of bytes written to the buffer is returned**

Introduction, part 2

- **To reduce the dynamic storage needed and transmitted by the PL/I IMS importers, we have introduced a radically new language feature**
- **Again, it is generally useful even outside of IMS**
- **This feature addresses the problem of “sparse” arrays**
- **This is RFE 29703 from Credit-Suisse**

Introduction, part 2

- **For example, 61204 bytes will be allocated for this structure - even though many of the varying strings might be much smaller than the maximum**

```
dcl
   1 xmit based(p) unal,
     2 dcount     fixed bin(31),
     2 data(dx refer(dcount)),
        3 name    char(100) varying,
        3 street  char(100) varying,
        3 city    char(100) varying;

 dcl p  pointer;
 dcl dx fixed bin(31);

 dx = 200;
 allocate xmit;
```

Introduction, part 2

- **You could change each of the CHAR(100) VAR fields into a POINTER, but**
- **1) then to assign a string to one of these fields, you would first have to allocate a based string and then do the assignment**
- **2) And perhaps more importantly, if the structure contains pointers, then you cannot usefully write it to a file or transfer it to another address space**
- **Using an OFFSET instead of a POINTER helps with 2) but still leaves 1) as a problem**

LOCATES attribute

- **The new LOCATES attribute and associated built-in functions and pseudovariables help address these issues**
- **The LOCATES attribute essentially turns an OFFSET attribute into a typed offset (but the type must be a string type)**
- **And the LOCVAL built-in function and pseudovariable let you dereference it**

LOCATES attribute

- **For example, you could change the earlier declare to**

```
dcl
  1 xmit based(p) unal,
     2 dcount     fixed bin(31),
     2 data(dx refer(dcount)),
        3 name    offset(pool) locates( char(100) varying ),
        3 street  offset(pool) locates( char(100) varying ),
        3 city    offset(pool) locates( char(100) varying ),
     2 pool area(64000);
```

LOCATES attribute

- **Then to assign to the first name field, you could write**
  - **call locnewvalue('Sherlock Holmes', name(1), pool);**
- **But since NAME is declared as OFFSET(POOL), you could write the simpler**
  - **call locnewvalue('Sherlock Holmes', name(1));**
- **Or, you could let the compiler generate this call for you and write simply**
  - **name(1) = 'Sherlock Holmes';**

LOCATES attribute

- **So, to fill out the first data item, you could write**
  - **name(1) = ‘Sherlock Holmes’;**
  - **street(1) = ‘221B Baker Street’;**
  - **city(1) = ‘London’;**
- **This is exactly what you would write if no OFFSET was involved!**
- **So, to fill the whole array, you would not even need to know that there was any indirection going on**

LOCATES attribute

- **However, for other references to the fields, you would have to use the LOCVAL built-in function to dereference the offset. For example**
  - **put skip list(name(1));**
  - **if city(1) = ‘London’ then**
- **Would be rejected by the compiler as severe errors. You would have to write instead**
  - **put skip list(locval(name(1)));**
  - **if locval(city(1)) = ‘London’ then**
- **etc**

LOCATES attribute

- **But, if the OFFSET has an implicit qualifying area, then LOCVAL can be omitted when the OFFSET is used as the (first) argument to**
  - **INDEX(R), SEARCH(R), VERIFY(R)**
  - **LENGTH**
  - **MAXLENGTH**
  - **TALLY**
  - **TRIM**
- **MAXLENGTH will always be a constant**
- **For all the other functions, if the offset is zero, then the result would be zero**

LOCATES attribute

- **Note that the statement**
  - **call locnewvalue('Sherlock Holmes', name(1));**
- **Will get only enough room from the area to hold the new value – this is great if the string is written to only once**
- **If you want to get enough space to hold the maxlength, you should use the LOCNEWSPACE function and then use the LOCVAL pseudovariable. E.g.**
  - **call locnewspace(name(1));**
  - **locval(name(1)) = 'Sherlock Holmes';**

LOCATES attribute

- **If you compare this declare to its original, you might notice that it uses more storage than the original**

```
dcl
  1 xmit based(p) unal,
     2 dcount     fixed bin(31),
     2 data(dx refer(dcount)),
        3 name    offset(pool) locates( char(100) varying ),
        3 street  offset(pool) locates( char(100) varying ),
        3 city    offset(pool) locates( char(100) varying ),
     2 pool area(64000);
```

LOCATES attribute

- However, in the IMS importer scenario (and other scenarios), the data is assigned once and then never changed
- Hence when the structure is transferred to another address space (or written to a file), only the currently used part of the area is important
- In particular, the amount of storage that needs to be transported is given by the simple formula
  - loc(pool) + cstg(pool)
- Although we could optimize this calculation if we introduced a new built-in function to do it

LOCATES attribute

- **Also, if you look at this structure, you might ask how big should the area be and could the compiler calculate the needed size?**

```
dcl
  1 xmit based(p) unal,
     2 dcount     fixed bin(31),
     2 data(dx refer(dcount)),
        3 name    offset(pool) locates( char(100) varying ),
        3 street  offset(pool) locates( char(100) varying ),
        3 city    offset(pool) locates( char(100) varying ),
     2 pool area(64000);
```

LOCATES attribute

- For that purpose, we have introduced the last of the new built-in functions to support LOCATES, namely the LOCSTG function
- LOCSTG returns the number of bytes needed to hold all the storage required if all the elements using LOCATES had storage allocated
- For a BASED with REFER, the structure doesn't need to be allocated, but the compiler must be able to calculate the initial REFER values
- In our example, this means the DX value must be set
- And LOCSTG would return 61200

Increased Usability

Language enhancements

UTF-16 pictures

- **PICTURE numeric variables hold FIXED DEC and FLOAT values as CHAR**
- **The new WIDEPIC variables hold these values as WIDECHAR except**
  - **The picture specification must not contain any A or X (i.e. it must be numeric)**
  - **The picture specification must not contain any currency symbols**
  - **The picture specification must not contain any overpunch symbols**
- **WIDEPIC variables can be used in arithmetic calculations in the same way that the equivalent PIC variables can be used**

UTF-16 pictures

- **WIDEPIC allows PIC to become UTF-16 just as WIDECHAR did for CHAR**
- **so, given**
- **declare wp widepic'9V(4)9' init(3.1415)**
- **wp could be overlaid with WIDECHAR(5)**
- **wp holds the bytes '0033_0031_0034_0031_0035'wx**
- **This is RFE 27954 from Allianz**

INDEXR built-in function

- **The new INDEXR built-in function does for INDEX what SEARCHR bzw VERIFYR do for SEARCH bzw VERIFY**
- **Namely, INDEXR searches for the first occurrence of one string within another when searching from the right**
- **So while INDEX(‘Berg auf, Berg ab’, ‘Berg’) returns 1**
- **INDEXR(‘Berg auf, Berg ab’, ‘Berg’) returns 11**

DEFAULT statement

- **The DEFAULT statement has been expanded both to conform to the original PL/I standard, but more importantly to address the problems caused by statements such as**
  - **DEFAULT RANGE(I) INIT((*) ‘’);**
- **This statement wants to initialize any variable whose name starts with I to zero (or blanks if character or widechar)**
- **This is a perfectly good idea, except the following program will get an E-level compiler message:**

DEFAULT statement

```
talk: proc options(main);

  default range(i) init( (*) '' );

  display( test('31415') );

  test: proc( input ) returns( fixed bin(31) );
    dcl input char(*);
    dcl ix fixed bin(31);
    ix = input;
    return( ix );
  end;
end;
```

- **IBM2432I E 8.0 The attribute INITIAL is invalid with parameters and is ignored.**

DEFAULT statement

- **The DEFAULT statement in the original PL/I standard allows a more powerful predicate clause**
- **Rather than just the RANGE specification, you can specify a parenthesized logical expression using attribute keywords (including RANGE and the pseudo-attribute MEMBER)**
- **So now you could do what you could change this example to**
  - **DEFAULT (RANGE(I) & ^ INITIAL) INIT((*) ‘’);**

DEFAULT statement

- **This expanded statement is very powerful and could be used to describe the language defaults. For example, for FLOAT**
  - **DEFAULT (FLOAT & BIN) PREC(21);**
  - **DEFAULT (FLOAT & DEC) PREC(6);**
- **And for FIXED under RULES(IBM)**
  - **DEFAULT (FIXED & BIN) PREC(15);**
  - **DEFAULT (FIXED & DEC) PREC(5);**

ALLOCATE built-in function

- **The ALLOCATE built-in function now accepts an optional second argument which must specify an AREA from which to allocate the specified number of bytes**
- **So, while ALLOCATE(1000) obtains 1000 bytes from HEAP storage and returns a pointer to the allocated storage**
  - **if the requested storage is not available, the STORAGE condition is raised**
- **ALLOCATE(1000, X) obtains 1000 bytes from the AREA X and returns the offset into X of the allocated storage**
  - **If the requested storage is not available, the AREA condition will be raised**

CALL statement in macros

- **Given a preprocessor procedure such as**

```
%DCL QSUB ENTRY;
%QSUB: PROC;
   ANSWER('/* in QSUB */') SKIP;
%END QSUB;
```

- **You could invoke it in your code via the simple**

```
QSUB
```

CALL statement in macros

- **But if you tried to invoke it from another a preprocessor procedure such as**

```
%DCL PSUB ENTRY;
%PSUB: PROC;
   ANSWER('/* in PSUB */') SKIP;
   QSUB;
%END PSUB;
```

- **The preprocessor would object with an S-level message**

- **But since QSUB is a procedure (i.e. it does not RETURN anything), you should be able to invoke via a CALL statement**

CALL statement in macros

- **The 4.4 compiler now lets you do this**

```
%DCL PSUB ENTRY;
%PSUB: PROC;
   ANSWER('/* in PSUB */') SKIP;
   CALL QSUB;
%END PSUB;
```

- **This is RFE 32334 from Telcordia**

CANCEL THREAD

- **The new CANCEL THREAD statement does exactly that**
- **It lets you cancel a thread that you have attached**
- **This was a FITS requirement from La Caixa**

Miscellaneous user requirements

Recommending STATIC

- **Many user programs contain declares which are arrays or structures that are declared with INIT attributes and never changed. This is perfectly ok if the STATIC attribute is specified, but very bad if not**

- **For example**

```
DCL BVSFS (3,0:100,15)     FIXED DEC (3,2) INIT ((15) 1.00,
    (15) 1.00, (15) 1.00, (15) 1.00, (15) 1.00, (15) 1.00,
    (15) 1.00, (15) 1.00, (15) 1.00, (15) 1.00, (15) 1.00,
    (15) 1.00, (15) 1.00, (15) 1.00, (15) 1.00,
 0.94,0.95 ,0.95 ,0.95 ,0.95 ,0.96 ,0.96 ,0.95 ,0.95 ,0.95 ,0.95 ,0.95,
 0.95, 0.95, 0.95,
 0.88,0.89 ,0.89 ,0.90 ,0.90 ,0.91 ,0.91 ,0.91 ,0.91 ,0.91 ,0.91 ,0.90,
 0.90, 0.90, 0.91, . . .
```

Recommending STATIC

- **This will generate a vast amount of code that will be executed every time the containing procedure is called. This is terrible for run-time performance**
- **It also stresses the compiler under the optimize option because the compiler use a lot of time and space trying to optimize the thousands of generated assignment statements.**
- **This “variable” should be declared as STATIC**
- **Even better would be to declare it as STATIC NONASGN**

Recommending STATIC

- **The 4.4 compiler will now issue a W-level message for any declares**
  - **with more than 100 INITIAL items**
  - **but with a storage class other than STATIC**
- **This should help you identify possibly poor code**
- **Change it and you improve compile-time and run-time performance**
- **This is RFE 30159 from NY Life**

NOINCLUDE option

- **The new NOINCLUDE option specifies that the MACRO preprocessor must be invoked if there are any %INCLUDE or %XINCLUDE statements in the source**
- **The new INCLUDE option specifies that the final compiler pass can handle %INCLUDE and %XINCLUDE statements**
- **This matches the old OS PL/I compiler's INCLUDE option**
- **INCLUDE is the default**
- **This is RFE 31140 from Credit-Suisse**

NULLSTRPTR option

- **The NULLSTRPTR suboption of the DEFAULT option specifies via the SYSNULL bzw NULL suboption whether the assignment/comparison of '' to a pointer results in SYSNULL bzw NULL being assigned/compared to the pointer.**
- **The new suboption STRICT will cause such assignments and comparisons to be flagged as invalid**
- **The default remains NULLSTRPTR(NULL)**
- **This is RFE 28536 from Raiffeisen IT**

STMT option

- **When this program is compiled under 4.3 with the STMT option**

```
talk: proc( rc ) returns( char(6) );

  declare rc fixed bin;

  select( rc );

    when( 00 ) return( 'ok' );
    when( 04 ) return( 'warning' );
    when( 08 ) return( 'error' );
    when( 12 ) return( 'severe' );
    otherwise  return( '???' );
  end;
end;
```

STMT option

- **It produces a message that refers to stmt 7, but there is no stmt 7 in the listing!**

```
Compiler Source
  Line.File     Stmt
     1.0           1   talk: proc( rc ) returns( char(6) );
     2.0           2     declare rc fixed bin;
     3.0           3     select( rc );
     4.0           4       when( 00 ) return( 'ok' );
     5.0           6       when( 04 ) return( 'warning' );
     6.0           8       when( 08 ) return( 'error' );
     7.0          10       when( 12 ) return( 'severe' );
     8.0          12       otherwise  return( '???' );
     9.0          14     end;
    10.0          15   end;

Message     Statement  Message Description
IBM1185I W          7  Source in RETURN statement has length greater than
                       that in the corresponding RETURNS attribute.
```

STMT option

- **To help with this situation, under the STMT option the 4.4 compiler will include line.file numbers in the message listing (along with the statement number)**

- **This is RFE 35961 from Danske Bank**

STMT option

- **So, with the 4.4 compiler, the listing would look like**

```
Compiler Source
   Line.File     Stmt
      1.0           1   talk: proc( rc ) returns( char(6) );
      2.0           2     declare rc fixed bin;
      3.0           3     select( rc );
      4.0           4       when( 00 ) return( 'ok' );
      5.0           6       when( 04 ) return( 'warning' );
      6.0           8       when( 08 ) return( 'error' );
      7.0          10       when( 12 ) return( 'severe' );
      8.0          12       otherwise  return( '???' );
      9.0          14     end;
     10.0          15   end;

Message      Statement      Line.File  Message Description
IBM1185I W           7          5.0    Source in RETURN statement has length greater than
  that
                                       in the corresponding RETURNS attribute.
```

%INCLUDE in the listing

- **When the macro preprocessor is used to compile this program**

```
test: proc;

  %include file1;
end;
```

%INCLUDE in the listing

- **the beginning and end of the include are marked by comments in the listing**

```
Compiler Source
   Line.File
      1.0      test: proc;
      2.0
      3.0       /* BEGIN %INCLUDE FILE1 */
      1.1        dcl a1  fixed bin(15);
      2.1
      3.1        a1 = 17;
      3.0       /*   END %INCLUDE FILE1 */
      4.0      end;
```

%INCLUDE in the listing

- **But if the macro preprocessor is not used, the 4.3 listing is not so nice**

```
Compiler Source
   Line.File
      1.0      test: proc;
      2.0
      3.0        %include file1;
      1.1        dcl a1  fixed bin(15);
      2.1
      3.1        a1 = 17;
      3.0
      4.0      end;
```

%INCLUDE in the listing

- **But the 4.4 listing is much better (thanks to RFE 31139 from Credit-Suisse)**

```
Compiler Source
   Line.File
      1.0      test: proc;
      2.0
      3.0        %include file1;
      1.1      /*** Begin %include DD:SYSLIB(FILE1) ***/
      1.1        dcl a1  fixed bin(15);
      2.1
      3.1        a1 = 17;
      3.1      /*** End   %include DD:SYSLIB(FILE1) ***/
      3.0
      4.0      end;
```

Migration considerations

ARCH level

- **The compiler is itself compiled with ARCH(7)**
- **But it still supports generating code with the ARCH(6) option**
- **However, support for ARCH(6) is likely to end in the near future**

WIDECHAR and SQL

- **WIDECHAR host variables will now always be assigned a codepage of 1200**
- **The CCSID0 and NOCCSID0 option will no longer have any effect on the codepage assigned to WIDECHAR host variables**
- **This change will probably be backfit to the 4.2 and 4.3 compilers**

Compilation return code changes

- **The message flagging declares that “should” be STATIC is a W-level message**
- **This means some compiles may end with a return code of 4 rather than 0**

Listing changes

- **If the STMT option is specified, the source listing will now include both the logical statement number and the source line-and-file numbers**

- **If the final compiler pass (rather than a MACRO pass), processes any %INCLUDE or %XINCLUDE statements, then the listing will now contain a comment at the start and at the end of the included source**

UTF-8 source

UTF-8 source

- **Almost all the RDz tools work with UTF-8**
- **This includes RTC – text files stored in RTC are stored as UTF-8**
- **But the Windows PL/I compiler works only with ASCII source**
- **So, when an RDz compile is done, the file is extracted from RTC and converted to ASCII – and the same is done for all include files**
- **This slows down the compile process, and it can also create problems if the wrong ASCII codepage is used**

UTF-8 source

- **The Windows PL/I compiler in RDz now supports UTF-8 source under the option ENCODING(UTF8) (with ENCODING(ASCII) as the default)**
- **Under ENCODING(UTF8), in MARGINS(n,m), the values n and m refer to counts of UTF-8 characters, not bytes**
- **So, if a source line contained pure ASCII except for one ä, then under MARGINS(2,72), bytes 2 through 73 would contain the compiler source**
- **Under ENCODING(UTF8), the NAMES option has to specify UTF-8 values for the lowercase and uppercase extralingual characters allowed**

Im Ueberblick

performance

- Faster listing generation
- More DFP exploitation in conversions from PIC
- Inlining of FIXED to WIDECHAR
- Other improvements to UTF-8 and UTF-16

middleware support

- Improved SQL support
  - More helpful messages from the SQL preprocessor
  - EMPTYDBRM option
  - Structures with arrays supported
  - Nicer commenting out of SQL statements

middleware support

- Improved IMS support
  - Base 64 encoding and decoding functions
  - XML normalization and cleaning functions
  - LOCATES attributes and associated functions for sparse arrays

usability

- WIDEPIC attribute
- INDEXR built-in function
- DEFAULT statement expanded
- ALLOCATE n bytes from an AREA
- CANCEL THREAD statement
- Nicer handling of INCLUDEs
- Compiler will recommend when to change dcl's to STATIC

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