Enterprise COBOL Unicode and XML Support

Enterprise COBOL Unicode and XML Support - Course Objectives

On successful completion of this class, the student, with the aid of the appropriate reference materials, should be able to:

1. Describe the attributes of Unicode, and explain the difference between the three formats of Unicode data (UTF-8, UTF-16, UTF-32)

2. Code Unicode data items, Unicode literals, and Unicode hex literals in a COBOL program

3. Use intrinsic functions to convert between code pages including EBCDIC, ASCII, and Unicode

4. Describe the basic rules for XML document structure

5. Invoke the IBM high speed XML parser from a COBOL program to extract data from an XML document into a COBOL record structure

6. Use the XML GENERATE statement to create XML data from a COBOL data structure

7. Use Enterprise COBOL 4.1 or later facilities to work with namespaces and attributes, and to parse XML documents a record or a segment at a time

8. Use Enterprise COBOL 4.2 or later to validate an XML document against a schema stored in an external file or an internal data item.

Enterprise COBOL Unicode and XML Support - Topical Outline

Day One

Day Two

Enterprise COBOL Unicode and XML Support - Topical Outline, 2

Day Two, continued

Acknowledgement: Special thanks to Tom Ross and Nick Tindall of IBM for their invaluable assistance.

This page intentionally left almost blank.

Section Preview

- **COBOL Support For Unicode**
  - **What is Unicode?**
  - **Unicode Support in Enterprise COBOL**
  - **When Will You Need To Use Unicode Support?**
  - **Things To Watch Out For**
  - **Set Up and Handling Unicode (Machine Exercise)**

What Is Unicode?

- **Unicode is a character encoding scheme designed to support all characters in all human written languages**
  - **Theoretically eliminating issues when trying to include characters from multiple languages on a single web page or other document**
    - ✗ Think: "Universal codepage"

- **The details are way beyond the scope of this discussion; however some good background information can be found at these locations on the Web:**
  - **The Unicode Consortium home page has the standards**
    - ✗ http://www.unicode.org
  - **IBM has a site discussing its support of Unicode**
    - ✗ http://www.ibm.com/developerworks/webservices/library/ws-codepages

What Is Unicode?, continued

❒ **Unicode character string data is encoded in one of three ways**

- **UTF-32 - every character is represented by a 32-bit integer (actually, only the rightmost 21 bits are used): four bytes per character**

- **UTF-16 - most characters are represented by a 16-bit pattern (two bytes)**

  ✗ However, some characters are represented by a pair of 16-bit patterns (total of four bytes) called surrogate pairs

  ✗ The Enterprise COBOL compiler does not recognize surrogate pairs as such

    ➢ For example, if a surrogate pair is found in a National string, COBOL will count the length as two Unicode characters instead of one

  ✗ There is a variation of UTF-16 called "UTF-16LE" (LE for Little Endian: bytes are stored least-significant first); COBOL does not support this format

    ➢ Some UTF-16 data begins with a pattern called a Byte Order Mark (BOM) that indicates if the data is big endian or little endian; COBOL does not support this, either

- **UTF-8 - each Unicode character is represented by one, two, three, or four bytes, depending on the character**

  ✗ The one-byte codes are essentially the basic ASCII characters, so in a sense most ASCII character strings are UTF-8

❒ **UTF stands for Unicode Transformation Format**

What Is Unicode?, continued

- **The general Unicode code points for UTF-32 are listed in an Appendix of this handout, for background information only**

- **We care about Unicode because Java, XML, and many other Web-related technologies require Unicode support - and we care about these technologies**

  - **The mainframe hardware instruction set has been enhanced to provide some Unicode support**

  - **Now COBOL on the mainframe supports Unicode**

    - ✗ As does PL/I, C, and, of course, Assembler

Unicode Support in Enterprise COBOL

❒ **First, as background, you need to know that before Unicode became widely adopted IBM was pushing the envelope by supporting its own standard, DBCS - Double Byte Character Set**

- ♦ **The need to support Japanese and other Asian languages drove the development of this in the 1980's**
- ♦ **Most IBM-supported programming languages support DBCS**
  - ✗ DBCS data may be embedded in classic, single byte data by providing shift-in (X'0E') and shift-out (X'0F') characters around the DBCS string
- ♦ **In COBOL, a PICTURE clause that includes G or N picture characters is considered to be DBCS character data**
- ♦ **With the advent of Unicode, DBCS support needs to be carried forward while at the same time adding support for Unicode**
  - ✗ Ultimately, DBCS will probably fade away
  - ✗ But IBM does not want to break existing code that relies on DBCS handling (there's a lot of it out there)

❒ **Single byte EBCDIC and, in some instances, DBCS can be used for forming COBOL words, literals, picture strings and comments**

- ♦ **But the compiler cannot compile source code written in ASCII or Unicode**

Unicode Support in Enterprise COBOL, 2

❒ **This compiler uses the term National to describe data coded in UTF-16**

- **A data item described as having USAGE NATIONAL will be assumed to require two bytes for every character in the corresponding PIC clause**
  - ✗ The PICTURE character to use is N
  - ✗ However, N is also sometimes used for DBCS data, as is G
- **If a data item is declared with no USAGE clause and the picture clause uses N's, it is ambiguous: is the data Unicode or DBCS?**
  - ✗ A compiler option is provided to remove ambiguity:

    NSYMBOL({DBCS|NATIONAL})
  - ✗ Note that using a PICTURE character of G is always unambiguous: it always represents DBCS data (sometimes called Graphic data)
- **There is another compiler option: {DBCS | NODBCS} where DBCS means the compiler should recognize shift-in and shift-out characters**
  - ✗ Setting NSYMBOL(NATIONAL) forces DBCS to be set also; and in V3R4 of the compiler, the supplied default is DBCS

Unicode Support in Enterprise COBOL, 3

❒ **Data can, of course, be read into a National data item**

- **You may also code National literals: bound a string of characters using N'...' or n'...' or N"..." or n"..."**
- **National literals may be used wherever a Display literal may be used:**
  - ✗ In a VALUE clause, for a National data item or National conditional value (level 88 item on an item of type National)
  - ✗ In the figurative constant ALL (*e.g.*: ALL N"*")
  - ✗ In a relation condition (*e.g.*: if last-name = n'Λατοσ' ...)
  - ✗ In the WHEN clause of a binary SEARCH
  - ✗ In the ALL, LEADING, FIRST, BEFORE, or AFTER phrases of INSPECT
  - ✗ In the DELIMITED BY phrase of STRING and UNSTRING
  - ✗ In DISPLAY and EVALUATE statements
  - ✗ As an argument for CALL … BY CONTENT, CALL … BY VALUE, INVOKE … BY VALUE
  - ✗ In method names
  - ✗ In the argument list to these intrinsic functions: DISPLAY-OF, LENGTH, LOWER-CASE, MAX, MIN, ORD-MAX, ORD-MIN, REVERSE, UPPER-CASE
  - ✗ In compiler-directing statements COPY, REPLACE, TITLE
  - ✗ As a sending item in INITIALIZE, INSPECT, MOVE, STRING, UNSTRING

Unicode Support in Enterprise COBOL, 4

❒ **Maximum length of a National literal is 80 characters (160 bytes)**

❒ **Now, suppose you used a National literal for a data item**

✗ For example:

```
01  Book-gr national pic n(6) value n'βιβλια'.
```

✗ Of course, you might have those Greek characters available while coding using English characters, but most likely not

✗ How to interpret the resulting bit patterns from the characters you key in is dependent on the codepage in effect when you key in the program

✗ You can tell the compiler what codepage you're using through another compiler option, CODEPAGE:

CODEPAGE(*ccsid_#***)**

✗ *ccsid_#* is a numbered EBCDIC code page (Coded Character Set IDentifier)

✗ The default is 1140 which is Latin-1 with the Euro symbol; some other values are listed on the following page

❒ **IBM has a web site with links to pdf files describing all their supported code pages (mind the wrap):**

http://www.ibm.com/servers/eserver/iseries/software/globalization/codepages.html

Unicode Support in Enterprise COBOL, 5

❒ **There are hundreds of CCSID values to choose from; some common samples:**

- ➢ 37 - Latin-1 (EBCDIC)
- ➢ 500 - International Latin-1 (EBCDIC)
- ➢ 819 - ASCII
- ➢ 1047 - Latin 1/Open Systems (EBCDIC)
- ➢ 1140 - Latin-1 w/ Euro (EBCDIC)
- ➢ 1143 - Finland, Sweden (EBCDIC)

♦ **Note that code page 1200 is UTF-16 and 1208 is UTF-8 (you cannot specify these in the CODEPAGE compiler option, but you may use them in some of the intrinsic functions that support codepage conversions)**

❒ **In any case, you certainly won't be able to do this:**

```
01  Book-jp national pic n(1) value n'本 '.
```

✗ To key in values not in the codepage you're using, you must use National hexadecimal literals

✗ NX"..." or NX' ... ', where the N and the X may be any case and the contents in the quotes must be the hex string representing the Unicode character you want

✗ Instead of the above code, you need something like this:

```
01  Book-jp national pic n(1) value nx'672C'.
```

Unicode Support in Enterprise COBOL, 6

❒ **Here are some small strings of characters and their corresponding representations in various code pages (top line is character string, all other values are hex):**

character string: `Here is data`

EBCDIC*: `C88599854089A2408481A381`

ASCII / UTF-8: `486572652069732064617461`

UTF-16:
`0048006500720065002000690073002000640061007400 61`

*** - note that all EBCDIC code pages encode English alpha-numeric characters, and many punctuation characters, the same**

♦ **There are 13 EBCDIC characters that vary across EBCDIC character map codepages but that must always be defined when using locale settings; here are some sample mappings:**

character:	[	]	{	}	!	\	^	~	`	$	|	@	#
EBCDIC 1140:	BA	BB	C0	D0	5A	E0	B0	A1	79	5B	4F	7C	7B
EBCDIC 500:	4A	5A	C0	D0	4F	E0	5F	A1	79	5B	BB	7C	7B
EBCDIC 1047:	AD	BD	C0	D0	5A	E0	5F	A1	79	5B	4F	7C	7B
EBCDIC 1143	B5	9F	43	47	4F	71	5F	DC	51	67	BB	EC	63
ASCII / UTF-8:	5B	5D	7B	7D	21	5C	5E	7E	60	24	7C	40	23

UTF-16:

[	]	{	}	!	\	^	~	`	$	|	@	#
005B	005D	007B	007D	0021	005C	005E	007E	0060	0024	007C	0040	0023

Unicode Support in Enterprise COBOL, 7

❒ **The compiler provides additional Unicode support as follows**

- **A MOVE from a DISPLAY item to a NATIONAL item will cause conversion from EBCDIC to UTF-16 automatically**
  - ✗ As usual, left justify and right truncate or pad; truncation is on two-byte units; padding is done with UTF-16 spaces (nx'0020')
  - ✗ Note: you are not allowed to MOVE a National item to a Display item (but see related intrinsic functions later)
- **Numeric integer (data items or literals) may be assigned to NATIONAL items and will be converted to Unicode numeric characters**
- **When reference modification is used for items defined as NATIONAL, both the starting location and length values represent the number of character positions, not the number of bytes**
- **Note that support for UTF-16 does not pay attention to surrogate pairs as such; that is, although a surrogate pair takes four bytes to represent one Unicode character, COBOL interprets a surrogate pair as two character positions**
  - ✗ The programmer is responsible for ensuring a surrogate pair is not split inappropriately

Unicode Support in Enterprise COBOL, 8

❒ **The compiler provides additional Unicode support as follows**

- **If a NATIONAL item or literal is compared to alphanumeric, display, DBCS, or numeric integer, the non-Unicode data is converted to UTF-16 for the comparison**

- **Note that comparisons are strictly binary, not cultural**

  ✗ That is, comparisons are simply done on the bit patterns, which may or may not be how the language would relate two items

    ➢ For example, the character Ä collates after `'z'` in Swedish, but after `'a'` in German

  ✗ To get cultural compares, you must use LE locale services

- **The RECORD KEY clause for VSAM KSDS files may be a NATIONAL data item**

  ✗ As may the ALTERNATE RECORD KEY (for alternate index support)

- **The FILE STATUS data item for any file may be NATIONAL category**

Unicode Support in Enterprise COBOL, 9

❒ **The compiler provides additional Unicode support as follows**

- ♦ **Figurative Constants, when used with National items…**

  - ✗ ZERO, ZEROS, ZEROES - one or more National zeros are used (NX'0030')

  - ✗ SPACE, SPACES - one or more National spaces are used (NX'0020')

  - ✗ HIGH-VALUE, HIGH-VALUES, LOW-VALUE, LOW-VALUES - generate NX'FFFF' and NX'0000' as you might expect (not supported until V3R4)

    - ➢ Note: do not mix DISPLAY and NATIONAL (Unicode) versions of these figurative constants (*e.g.*, comparisons, moves, *etc.*; this will cause conversion and surprising substitutions)

  - ✗ QUOTE, QUOTES - use one or more National quotes (NX'0022') or National apostrophes (NX'0027'), depending on the setting of the compiler option {QUOTE|APOST}

Unicode Support in Enterprise COBOL, 10

❒ **The compiler provides additional Unicode support as follows**

- **When a National item is DISPLAYed to the console, it is automatically translated from UTF-16 to EBCDIC using the codepage option at compile time**
  - ✗ If displayed to SYSOUT (the default) no conversion is done
  - ✗ To force conversion, use the DISPLAY-OF function
- **When data is ACCEPTed from the console into a National item, it is automatically converted from EBCDIC, using the compile time CODEPAGE setting, to UTF-16**
  - ✗ If accepted from a file (say, SYSIN, the default) no conversion is done
- **Note that if any literal or identifier in a STRING, UNSTRING, or INSPECT statement is National than all literals and identifiers in that statement must be National items**
  - ✗ If the TALLYING option is used for INSPECT or UNSTRING, the value returned is the number of 2-byte encoding units
  - ✗ If the POINTER option is used for STRING or UNSTRING, the value returned or used represents the number of 2-byte encoding units offset from the start
- **The COBOL SORT and MERGE verbs can use National data items for the sort / merge key fields**

Unicode Support in Enterprise COBOL, 11

❒ **The compiler provides additional Unicode support as follows**

- **Two intrinsic functions support explicit conversion between Unicode and another codepage:**

  ✗ **DISPLAY-OF(***national-item* [[,]*ccsid*]**)** - given UTF-16 data in, returns EBCDIC, ASCII, or UTF-8 data out, using the codepage indicated by *ccsid*

  ➢ For example, UTF-16 to EBCDIC:

```
     move function display-of(in-str, 1047) to out-str
```

will move contents of in-str**, say:**
`0048006500720065002000690073002000640061 00740061`

and convert into out-str**:**
`C88599854089A2408481A381`

  ✗ And UTF-16 to ASCII:

```
     move function display-of(in-str, 819) to out-str
```

will move in-str**:**
`0048006500720065002000690073002000640061 00740061`

and convert into out-str**:**
`4865726520697320646174 61`

Unicode Support in Enterprise COBOL, 12

❒ **The compiler provides additional Unicode support as follows, continued**

- ♦ **Two intrinsic functions support explicit conversion between Unicode and another codepage, continued:**
  - ✗ **NATIONAL-OF(***display-item* [[,]*ccsid*]) - given EBCDIC, ASCII, or UTF-8 data in (as indicated by *ccsid*), returns UTF-16 data out
    - ➢ For example, EBCDIC 1047 to UTF-16:

```
move function national-of(desc, 1047) to out-desc
```

will move the contents of desc**, say:**

```
C88599854089A2408481A381
```

and convert it into UTF-16 in out-desc**:**

```
0048006500720065002000690073002000640061007400610
```

❒ **For both functions, the default** *ccsid* **is that specified in the CODEPAGE compiler option**

- ♦ **Which must represent an EBCDIC code page**

❒ **The NUMVAL and NUMVAL-C intrinsic functions can take National data in their arguments**

Unicode Support in Enterprise COBOL, 13

Converting data to / from Unicode

```
   01  Uni-data       pic N(20) national.
   01  Ebcdic-data    pic X(20).
   01  U8             pic X(20).
   01  DBCS-data      pic G(20) display-1.
   .
   .
   .
❶     move Ebcdic-data to Uni-data
❷     move function National-of(Ebcdic-data) to Uni-data

❸     move function National-of (U8, 1208) to Uni-data
❹     move function National-of (DBCS-data, 1399)
                to Uni-data

❺     move function Display-of(Uni-data) to Ebcdic-data
❻     move function display-of(Uni-data, 1208) to U8
❼     move function display-of(Uni-data, 1399)
                to DBCS-data
```

- **Notes**
  - ✗ Statements ❶ and ❷ both do the same thing: copy data from Ebcdic-data into Uni-data, converting it to UTF-16 based on the current codepage setting
  - ✗ Statement ❸ converts UTF-8 data to UTF-16, placing it into Uni-data
  - ✗ Statement ❹ converts Japanese EBCDIC data (CCSID 1399) to UTF-16, placing the result into Uni-data
  - ✗ Statement ❺ converts UTF-16 in Uni-data to EBCDIC
  - ✗ Statement ❻ converts UTF-16 in Uni-data to UTF-8 in U8
  - ✗ Statement ❼ converts UTF-16 in Uni-data to Japanese EBCDIC in DBCS-data

Unicode Support in Enterprise COBOL, 14

Converting between EBCDIC and ASCII

```
   01  EBCDIC-CCSID  pic 9(4) binary value 1140.
   01  ASCII-CCSID   pic 9(4) binary value 819.

   01  Uni-data      pic N(80) national.
   01  Ebcdic-data   pic X(80).
   01  ASCII-data    pic X(80).
   .
   .
   .
❶    move function National-of(Ebcdic-data, EBCDIC-CCSID)
            to Uni-data
     move function Display-of(Uni-data, ASCII-CCSID)
            to ASCII-data

❷     move function Display-of
           (function National-of
             (Ebcdic-data, EBCDIC-CCSID)), ASCII-CCSID)
              to ASCII-data
```

- The two statements at ❶ are equivalent to the single statement at ❷
- The reverse process also works
- Note that it is probably faster to just use INSPECT … CONVERTING

❒ The compiler provides additional Unicode support as follows

- The LENGTH intrinsic function of a National data item returns the length of the item in National characters
- The LENGTH OF special register of a National data item returns the length of the item in bytes

Unicode Support in Enterprise COBOL, 15

❒ **Enterprise COBOL V3R4 expanded UNICODE support further, to solve some problems that existed in earlier versions and to come closer to the 2002 standard for internationalization**

❒ **Group items in COBOL often work differently than elementary items**

- ♦ **Especially in moves and compares, but elsewhere also**

For example

```
01  Country-info.
    02  country-name      pic N(25).
    02  country-capitol   pic N(25).
01  Country-hold.
    02  hold-name         pic N(25).
    02  hold-capitol      pic N(30).
.
.
.
           move country-info to country-hold
```

- ♦ **Group moves act upon the single group item without respect to elementary items in the group**
  - ✗ **hold-name** will end up with the value of **country-name**
  - ✗ **hold-capitol** will have the 25 characters (50 bytes) of **country-capitol** followed by 10 bytes of EBCDIC spaces (not 5 Unicode spaces)
    - ➢ Even if you add USAGE NATIONAL at the group levels!

Unicode Support in Enterprise COBOL, 16

❒ **Group items in COBOL often work differently than elementary items, continued**

Another example

```
01  Country-info.
    02  country-name     pic N(25).
    02  country-capitol  pic N(25).
01  language             pic N(25).
01  summary-string       pic N(75).
. . .
           string country-info delimited by size
                  language delimited by size
                  into summary-string
```

- **Country-info is treated as an alphanumeric (classic) group, even though its elementary items are all national**

  ✗ This fails at compile time because it is not allowed to have both alphanumeric and national items in a STRING statement

Still another example

```
01  Country-info.
    02  country-name     pic N(25).
    02  country-capitol  pic N(25).
.
.
.
           inspect country-info tallying ctr
             for leading spaces
```

- **Looks for EBCDIC spaces in group level items**

Unicode Support in Enterprise COBOL, 17

❒ **So to handle these (and other) cases, the group level specification was added**

GROUP-USAGE [IS] NATIONAL

❒ **When this is placed at the group level, padding for group level MOVEs, comparisons for group level INSPECTs, and so on, use National characters**

- **Furthermore, all items in the group are now NATIONAL category (may not have non-Unicode data in a group designated with GROUP-USAGE NATIONAL)**
  - ✗ Do not specify a regular USAGE on a group item that has GROUP-USAGE clause or on any subordinate elementary item
  - ✗ Any subordinate signed numeric items must have SIGN IS SEPARATE clause
  - ✗ Any group that is defined without a GROUP-USAGE NATIONAL clause is an alphanumeric group, even if all the elementary items in the group are declared as NATIONAL

For example:

```
01  Country-info group-usage national.
    02  country-name     pic N(25).
    02  country-capitol  pic N(25).
.
.
.
           inspect country-info tallying ctr
             for leading spaces
```

- **Works as you would like it to, tallying Unicode spaces**

Unicode Support in Enterprise COBOL, 18

- **In general: USAGE NATIONAL at the group level causes subordinate groups and elementary items to act as alphanumeric groups and items when the group is specified in a verb:**
  - **Group moves / compares are byte-wise**
    - ✗ If one operand of a compare is a USAGE NATIONAL and the other is an alphanumeric literal, say, the literal will not be converted to National
    - ✗ Whereas if you compare a National elementary item to an alphanumeric literal, the literal will first be converted to National
  - **Group level INITIALIZE on USAGE NATIONAL group item is treated as an alphanumeric INITIALIZE**
  - **{ MOVE | ADD | SUBTRACT } CORRESPONDING on a group item defined with USAGE NATIONAL treat the subordinate items as alphanumeric items, and no conversion is done**
  - **A USAGE NATIONAL group, if used as a DB2 host variable, is treated still as alphanumeric**
  - **XML GENERATE from a USAGE NATIONAL item will treat the group as alphanumeric (discussed later)**
- **Using GROUP-USAGE NATIONAL at the group level eliminates those surprises**
  - **Note: cannot use JUSTIFIED for a GROUP-USAGE NATIONAL group**

Unicode Support in Enterprise COBOL, 19

❒ Compare behavior of a group level National USAGE to a GROUP-USAGE NATIONAL clause; that is:

```
01  Country-info usage national.
    02  country-name     pic N(25).
    02  country-capitol  pic N(25).
01  Country-hold.
    02  hold-name        pic N(25).
    02  hold-capitol     pic N(30).
.
.
.
           move country-info to country-hold
```

- ♦ Will still pad with trailing EBCDIC spaces to hold-capitol, since Country-info is still considered an alphanumeric group(!)

```
01  Country-info group-usage national.
    02  country-name     pic N(25).
    02  country-capitol  pic N(25).
01  Country-hold.
    02  hold-name        pic N(25).
    02  hold-capitol     pic N(30).
.
.
.
           move country-info to country-hold
```

- ♦ Will pad with trailing Unicode spaces to hold-capitol

Unicode Support in Enterprise COBOL, 20

❒ New National data types

- **Before version 3.4, only Unicode character strings (picture characters of N) were supported**

- **Now (Enterprise COBOL 3.4 and later), several new data types are provided:**

  ✗ **National-edited** - specify USAGE NATIONAL but allow B (for Unicode blank), 0 (for Unicode zero), and / (for Unicode slash) as well as [at least one] N for Unicode character:

```
05  account_no  pic nn/nn/nnnn  national.
```

  ✗ **National decimal** - specify USAGE NATIONAL but allow 9 (for Unicode numeric digit), V (for implied decimal place), P (for decimal scaling) and S (for Unicode sign); must have at least one '9'; if signed, **must** have SIGN [IS] {LEADING | TRAILING} SEPARATE [CHARACTER]:

```
05  un-price  pic s9(5)V99  national
       sign is leading separate.
```

Unicode Support in Enterprise COBOL, 21

❒ **New National data types, continued**

✗ **National numeric-edited** - specify USAGE NATIONAL but allow B (for Unicode blank), P (for scaling), for Z (to indicate suppression of leading non-significant zeros), comma (,) or period (.) or slash (/) (as Unicode insertion characters) possibly one of + - CR DB (as Unicode insertion characters indicating sign value) and possibly a currency indicator (floating or fixed asterisk (*), dollar sign ($) or other currency symbol as specified in the special-names paragraph)

```
05  out-price  pic $$,$$9.99         national.
05  balance    pic 99,999,999.99DB   national.
05  u-date     pic 99/99/9999        national.
05  no-widgets pic zz,zz9            national.
```

✗ **National floating-point** - specify USAGE NATIONAL but use a floating point format: {+ | - }*mantissa*E{+ |-}99 (if a sign is missing it is assumed to be a plus sign; *mantissa* must contain '9's representing decimal positions and either a period (.) to represent an actual decimal place or a V to represent an implied decimal place:

```
05  in-factor  pic 1.34E12  national.
```

❒ **National decimal and National floating-point may participate in the same arithmetic operations other numeric data types can appear in (ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPUTE, arithmetic intrinsic functions, comparisons,** *etc.***)**

Unicode Support in Enterprise COBOL, 22

❒ **Currency sign clause still works only with alphanumeric literals:**

Example

♦ **This code**

```
Environment division.
Configuration section.
Special-names.
    currency 'Eur' picture symbol '%'
    currency x'9f' picture symbol '$'.
.
.
.
data division.
working-storage section.

01  amount-field pic s9(7)v99 packed-decimal
        value +346928.33.
01  disp-1  pic %z,z99,999.99   national.
01  disp-2  pic $z,z99,999.99   national.
01  disp-3  pic %%%,%99,999.99  national.

Procedure division.

    move amount-field to disp-1, disp-2, disp-3
    display function display-of(disp-1 1140)
    display function display-of(disp-2 1140)
    display function display-of(disp-3 1140)
    goback.
```

♦ **Produces this on the SYSOUT file:**

```
Eur  346,928.33.
€  346,928.33.
   Eur346,928.33.
```

When Will You Need To Use Unicode Support?

❒ **Many of the conversions between Unicode and EBCDIC are handled by compiler generated routines and other processes**

- ♦ **For example, when using the DB2 coprocessor (compile option SQL in effect) the codepage CCSID is automatically coordinated between COBOL and DB2**

- ♦ **Java:COBOL interoperability uses Unicode implicitly "under the covers"**

❒ **But there may be times for you to explicitly use Unicode support**

- ♦ **Since Java is based on Unicode, when COBOL programs and Java methods are communicating, Java strings are in Unicode, so may want / need to use Unicode support**

- ♦ **XML documents and XHTML pages may be coded in UTF-16, UTF-8, ASCII, or EBCDIC**

  - ✗ You can parse XML documents encoded in EBCDIC or UTF-16 directly from a COBOL program

  - ✗ For ASCII or UTF-8, you can use National-of to convert to UTF-16 then parse, or use Display-of to convert the UTF-16 to an EBCDIC codepage then parse

- ♦ **There may be other applications for Unicode depending on your environment and set up**

Things To Watch Out For

- **If you convert the encoding of an XML document or HTML or XHTML page, you need to watch out for embedded information that is no longer valid**
  - **For XML, an** encoding="utf-8" **or** encoding="utf-16" **attribute may need to be changed**
  - **For HTML, a meta statement including something like** charset=utf-8 **or** charset=utf-16 **may need to be changed**
  - **XHTML might have either of these**
- **In all these cases, the** encoding **or** charset value **needs to be changed to reflect the new encoding scheme**
  - **First you need to see if such embedded information is present and if so, to change it, using something like this**
    - ✗ Here we assume you have converted UTF-8 data to EBCDIC on the way to converting to UTF-16:

```
01  utf8-char    pic x(16) value 'charset=utf-8"> '.
01  utf16-char   pic x(16) value 'charset=utf-16">'.
01  ebcdic-work  pic x(102).
.
.
.
           inspect ebcdic-work replacing all
              utf8-char by utf16-char
```

- **It is tricky to code for the general case; in many cases you can get by with ignoring this (if external sources, such as message headers, will be determining the document encoding, for example)**

Things To Watch Out For, 2

- **Also be careful to do this checking at the right point in time in your logic**
  - **For example, you may not be able to check for the presence of** charset=ascii **of a data item currently encoded in ASCII**
    - ✗ You need to get the data item into the same code page as your compile time CODEPAGE value first, so literal values are correctly interpreted
- **Note that you may need to convert data in a code page into Unicode (UTF-16) (using** function National-of()**) and then into the target code page (using** function Display-of()**) as in our example on page 22**
  - **That is, UTF-16 may need to be used as an intermediate stop, even if you do not intend to end up there**
- **The DISPLAY-OF and NATIONAL-OF functions output substitution characters when an input character has no corresponding output character in the respective code pages**
  - **DISPLAY-OF uses x'3F' for EBCDIC input, x'7F' for ASCII input, x'1A' for UTF-8 input, and x'001A' for UTF-16 input**
  - **NATIONAL-OF uses x'001A' for a substitution character**
- **If either conversion fails, a severe runtime error occurs (this is usually because Unicode conversion services have not been installed properly (or at all))**

Computer Exercise: Set Up and Handling Unicode

Lab Set Up

Run the rexx exec called D705STRT; this creates three libraries for you:

<*userid*>**.TR.CNTL** - contains JCL you will need to compile, link, and test the labs

<*userid*>**.TR.COBOL** - contains some starter code for later; this is where you will code your programs

<*userid*>**.TR.LOAD** - used to hold load modules; the JCL is set up to compile and link into this library, then run your programs from this library.

This also creates an empty flat file for use later in this lab:
<*userid*>**.TR.UTF16**

To run the exec, use ISPF 6 (command); and key in the following:

===> ex '_____.train.library(d705strt)' exec

and press <Enter>

This will run the rexx exec, which prompts you for a high level qualifier to use for the data set names mentioned above, defaulting to your TSO id; this is normally fine, so just press <Enter>. You should see a screen telling you the setup was successful.

Actual Lab

We have an HTML file coded in utf-8 that contains a Japanese kanji character in the first two bytes of the description field in each record. Our goal is to convert this file to utf-16 encoding in a file.

The big picture: for each record in the input file we want to …

* read the record into utf-8-rec
* display this record (will not be very readable)
* convert the utf-8 record to utf-16 into utf-16-rec
* convert the contents of utf-16-rec to ebcdic (code page 1140) into the field called ebcdic-work
* if the contents of ebcdic-work contains **charset=utf-8">** then convert that to **charset=utf-16">**
* display the contents of ebcdic-work
* write the contents of utf-16-rec to our output file

The file name of the input file is _____.TRAIN.HTMLUJ2. The file name of the output file will be <hlq>.TR.UTF16.

We have supplied skeleton code, named COBUNI in your TR.COBOL library; the member D705RUN1 in your TR.CNTL library is JCL to compile, link, and run COBUNI. The source for COBUNI is on the following pages.

The steps:

0. **[optional]** - download the utf-8 file in binary and open it in your browser
1. modify the code supplied to accomplish the tasks listed above; compile and run the code until successful
2. **[optional]** - download the utf-16 file in binary and open it in your browser

Note: you may need to rename the files on your PC to end in **.html**

Code Supplied As COBUNI

```
 Id division.
 Program-id. COBUNI.
* Copyright ©) by Steven H. Comstock, 2004       Ver 2

 Environment division.
 Input-output section.
 File-control.
     Select utf8in assign to utf8in.
     Select utf16 out assign to utf16out.

 Data division.
 File section.
 FD  utf8in
     recording f.
 01  utf-8-in        pic x(102).

 FD  utf160ut
     recording f.
 01  utf-16-out      pic n(102).

 Working-storage section.
 01  utf-8-rec       pic x(102).
 01  utf-16-rec      pic n(102).
 01  utf-16-work     pic n(204).
 01  ebcdic-work     pic x(102).
 01  utf8-char       pic x(16) value 'charset=utf-8"> '.
 01  utf16-char      pic x(16) value 'charset=utf-16">'.
 01  cntr            pic s9(4) binary value 0.
 01  Flags.
     02  end-of-file  pic x value '0'.
         88  end-in  value '1'.
```

Note: depending on how the compiler is installed in your installation, you may need to add a PROCESS statement at the front with NSYMBOL(NATIONAL) to get the program to compile correctly.

Code Supplied As COBUNI, 2

```
 Procedure division.
 start-up.
     display 'Starting program ...'
* open file and build document
     open input utf8in
          output utf16out
     perform get-in


     perform until end-in

        display 'Original record in utf-8: ' utf-8-rec


* convert utf-8-rec to utf-16 in utf-16-rec
```

```
* convert contents of utf-16-rec to ebcdic
*                                        in ebcdic-work
```

```
* display ebcdic-work contents
```

Code Supplied As COBUNI, 3

```
* for records with "charset" field, change value;
* that is: move 0 to cntr
*          inspect ebcdic-work tallying cntr for all
*                                              utf8-char
*         if cntr > 0
*            inspect ebcdic-work
*                    replacing all utf8-char
*                               by  utf16-char
*            display the resulting contents in
*                                            ebcdic-work
*            then convert the contents of ebcdic-work
*                    into utf-16 in utf-16-rec
*               (hint: use National-of function)
*         end-if
*
*************************************************
```

```
* write out utf-16-rec, get next input record
        write utf-16-out from utf-16-rec
        perform get-in

      end-perform


     display 'Ending program ...'

     close utf8in, utf16out
     goback.

 get-in.
     read utf8in into utf-8-rec
       at end set end-in to true
     end-read
```