

Unicode! Why do we need that?

SUG 08: Dave Evans
Tuesday 16:05



Agenda

- What Unicode is, and relationship to ASCII
- Code points, rendering, character sets, EBCDIC
- New User Language primitive type: Unicode
 - Intrinsic functions to convert between it and EBCDIC, and their implicit (under the covers) use
- UTF-8 and UTF-16 byte stream encodings
 - Intrinsic conversions between them and Unicode
- CharacterTranslationException class
- Other new (mostly intrinsic) functions



Agenda (*continued*)

- XML changes
 - Not yet coded; this was driving force for Unicode, but final Unicode design led to "standalone" features previously described
 - Strings (i.e., values, names, prefixes, URIs) in XmlDocument stored in Unicode
 - Most arguments and many results are Unicode
 - Deserialization (e.g., LoadXml) options to let your application choose how and when to avoid character translation exceptions



Agenda (*continued*)

- Edge case release boundary issues (only affect XML API)




Unicode - it is or will be ubiquitous

- Just as TCP/IP, XML, HTTP are ubiquitous
- For example, Unicode used in:
 - XML
 - HTML 4.0
 - .Net



Code points, character set mappings


- A code point represents a character: duh™
- Some characters are simple to deal with; common EBCDIC <-> ASCII (code points < X'80')
 - X'40' <-> X'20' (space)
 - X'F0' <-> X'30' (zero)
 - X'C1 <-> X'41' (uppercase A)
 - X'81 <-> X'61' (lowercase A)
- Upload SUG\$.TXT 
 - View with good old \$EbcDic

ASCII code points > X'7F'

- Upload SUGPND.TXT - Pound (sterling) currency
- ☘ ● \$Ebcdic: it & \$Ascii no error checks
- ☘ ● View with AsciiToEbcdic intrinsic function
- First part of Unicode project - map all but 32 of the 256 ASCII characters to EBCDIC
- Fixed clearly bad translations
 - ▶ <http://publibfp.boulder.ibm.com/epubs/pdf/dz9zs000.pdf> - yellow card "Latin/1 Open Systems"
- A few edge case incompatibilities
- Can customize your translation tables (but shouldn't need to)



ASCII code points > X'7F' (*continued*)

- Alan embedding editor: GUI for "special" chars
 - <http://www.kevinroth.com/rte/demo.htm> 
- Hex: A2: ¢
- Hex: A3: £
- Hex: A5: ¥ Yuan or Yen: ISO 4217: USD, JPY, CNY
- Hex: A8: ¨
- Hex: A9: ©
- Hex: AB: «
- Hex: AE: ®
- Hex: B0: °

ASCII code points > X'7F' (*continued*)

- Hex: B1: ±
- Hex: B2: ²
- Hex: B3: ³
- Hex: B5: μ
- Hex: B7: ·
- Hex: BB: »
- Hex: BC: ¼
- Hex: BF: ¿
- Hex: C0: À
- Hex: C1: Á



Sirius Software, Inc.

Need more than 256 characters: Unicode!

- ASCII inadequate for multi-lingual operation (e.g., proper name) and other symbols (™ U+2122)
- Upload SUGSMILE.TXT (save as Unicode Text)
Note: cancellation has implicit UnicodeToEbcDic
- Unicode 1.0 Oct 91, 5.1 Apr 08
- Convention U+hhhh, U+hhhhh, U+hhhhhh
- 1,114,112 code points (over 20 bits = 1,048,576)
U+hhhh: Basic Multilingual Plane (63,486 = 65,536 - 2,048 - 2; 2,048 = X'E000'-X'D800')
Surrogate = used for U+hhhhh/h



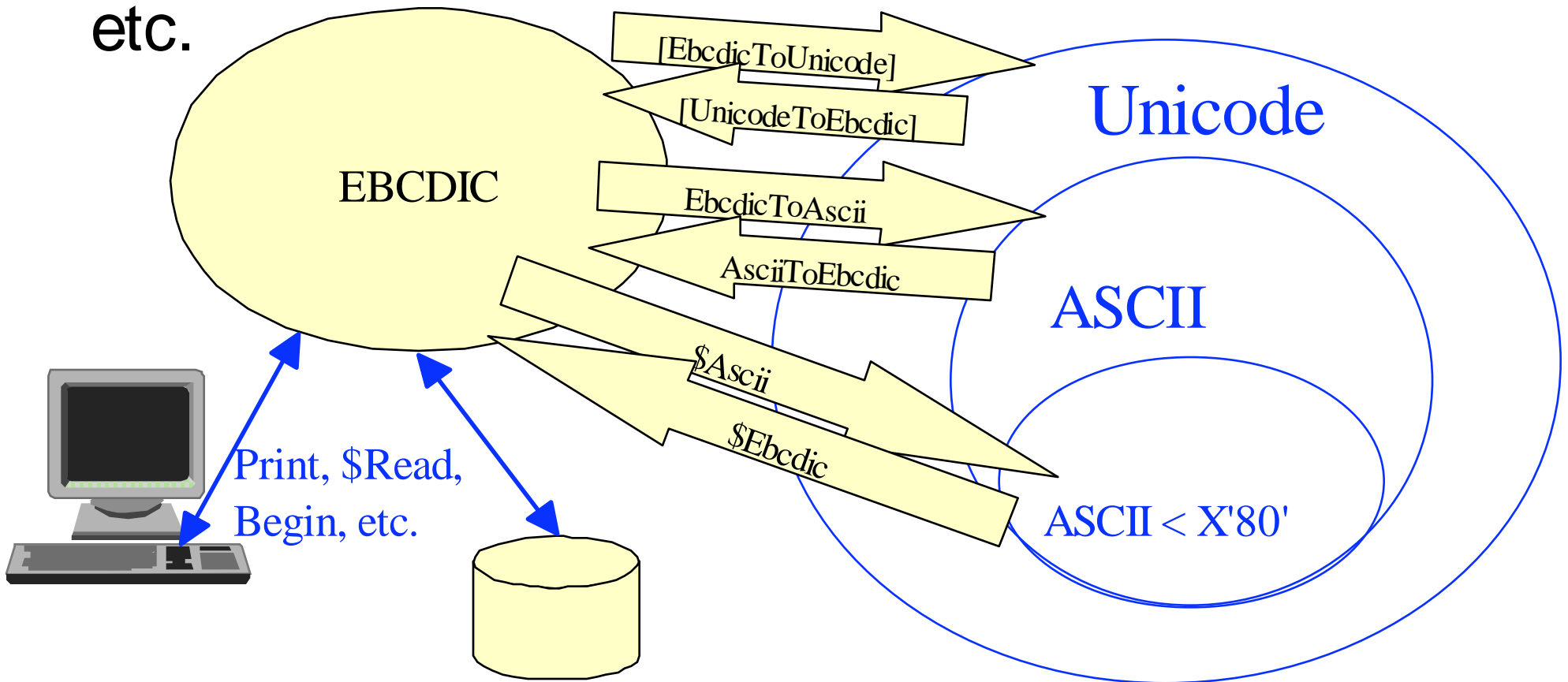
Need more ... Unicode! (*continued*)

- X'F900'-X'E000' = 6,400 Private use in BMP;
131,068 Private use outside BMP
 - Private use by Microsoft Webding
 - ▶ <http://www.fileformat.info/info/unicode/font/webdings/nonunicode.htm>
- <http://unicode.org/versions/Unicode5.1.0/>
 - ...unicode.org/Public/5.0.0/ucd/UnicodeData.txt
- Sirius Mods 7.3 has new Unicode "primitive" type
 - Can hold any BMP character
 - Sorry, no musical notation, nor Ugaritic (in Syria) and some other ancient lost languages

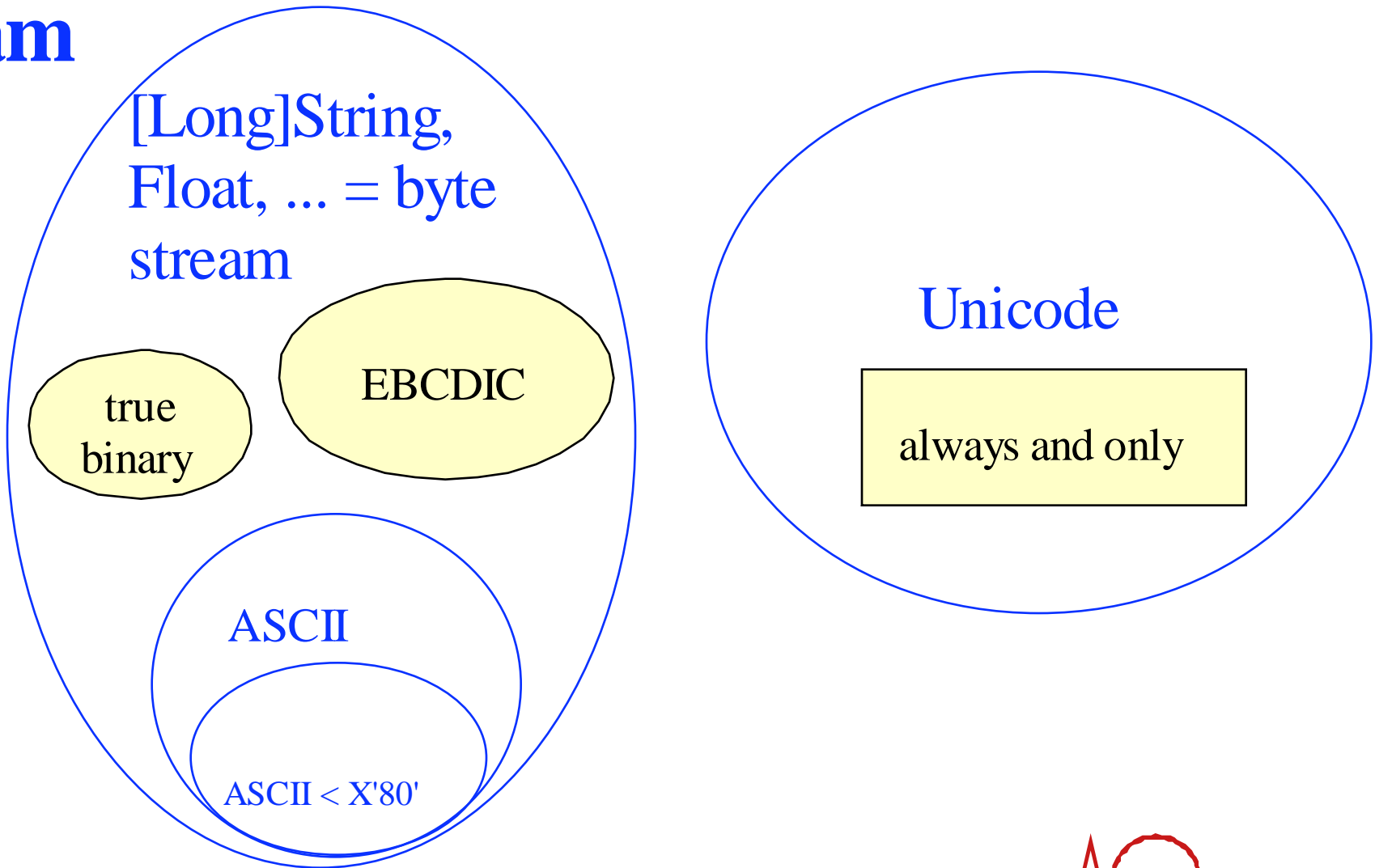


However, EBCDIC only has 256 characters

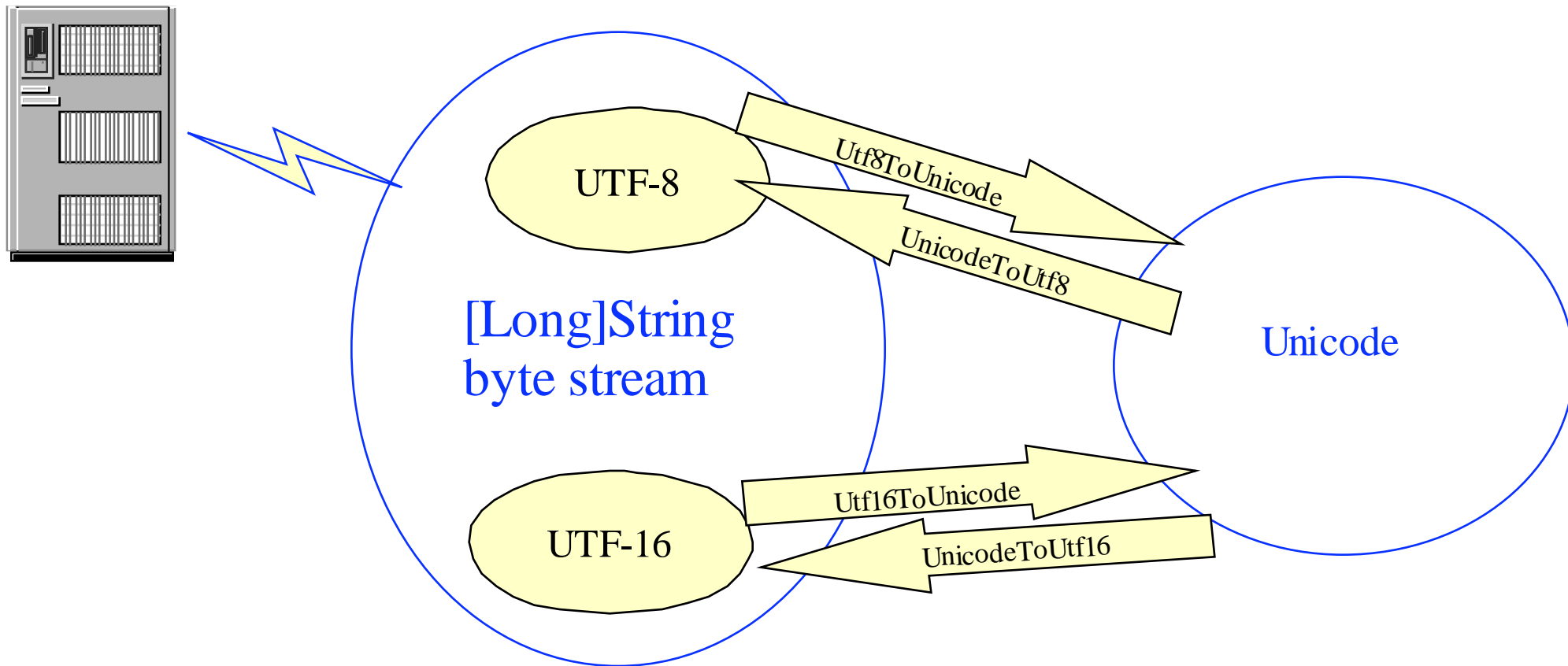
- EBCDIC used for 3270 and UL literals
- Note implicit conversion functions: assignment, etc.



Unicode: User Language type; EBCDIC/ASCII merely usage of byte stream



Unicode can be UTF-8/16 encoded, especially for network



UTF encode/decode (intrinsic) functions

- As shown, contained in [Long]String, presumed to have proper format
- UTF-16: direct map to Unicode representation
 - 2 bytes per character (for BMP characters)
 - also a good way to view Unicode character or encode Unicode literal
- ▶ Use the nice character diagrams in Unicode book
- ▶ ...:9204/unic.ul; modify: try non-BMP (see SirScan);
- ▶ show with hex
- ▶ Content-type:
<http://www.w3.org/International/O-charset>



UTF encode/decode (intrinsic) functions

(continued)

- UTF-8: most common Unicode network format
 - 1-3 bytes (for BMP characters) or 4 bytes per character
- To impress your friends (Unicode guys are clever):
 - "First portion(s)" of UTF-8 and UTF-16 representation must be "significant", i.e., each character has exactly one UTF-8 and one UTF-16 encoding
 - Can use byte operations (like PositionOf) to find any UTF-8 character in a stream (and similarly for UTF-16)
 - Given any byte in a UTF-8 or UTF-16 stream, you can find the beginning of the character using that byte



Sirius Software, Inc.

Untranslatable codes (EbcDicTo..) and characters (..ToEbcDic)

- \$Ascii and \$EbcDic: all codes translate (sometimes badly)
- EbcDicToAscii: 30 untranslatable **codes**
 - (intrinsic) function, throws CharacterTranslationException
- AsciiToEbcDic: 32 untranslatable **characters**
 - (intrinsic) function, throws CharacterTranslationException
 - CharacterEncode=True uses &#xhh; to encode




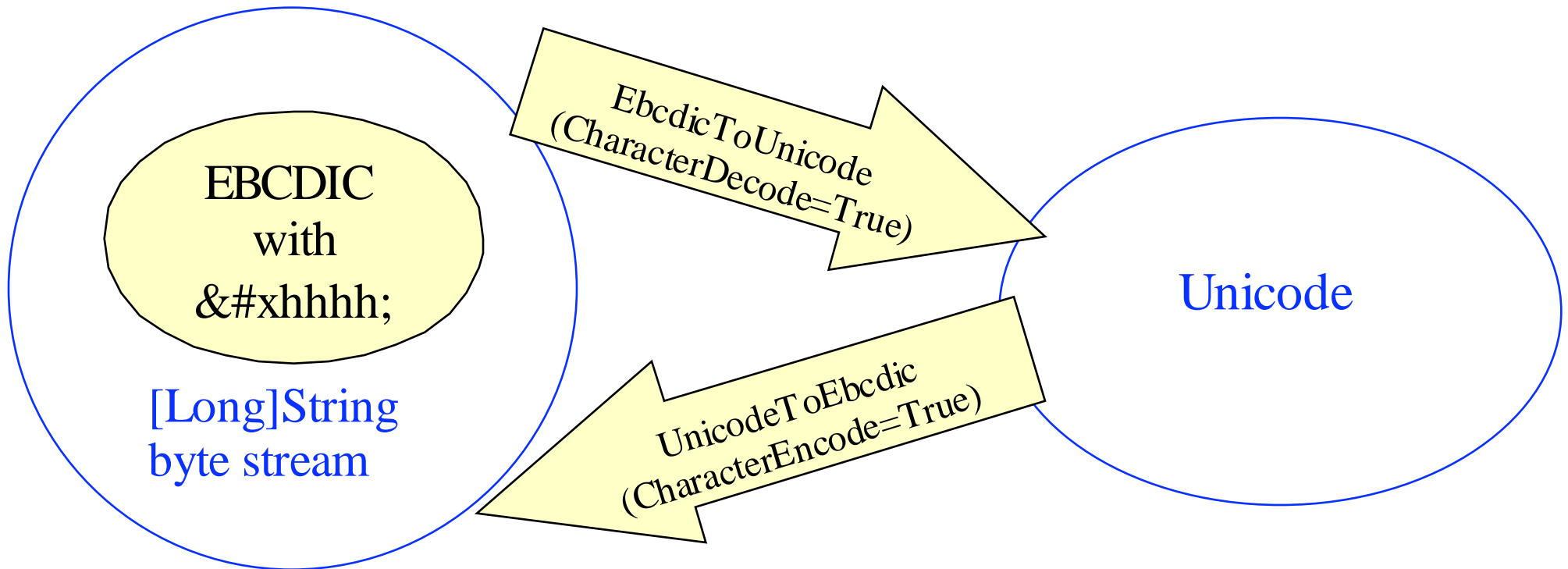
Untranslatable codes (EbcDicTo..) and characters (..ToEbcDic) (*continued*)

- EbcDicToUnicode: 30 untranslatable codes
 - (intrinsic) method, throws CharacterTranslationException
- UnicodeToEbcDic: over **1M** untranslatable **characters**
 - (intrinsic) function, throws CharacterTranslationException
 - CharacterEncode=True uses &#xhhhh; to encode




Character reference encoding/decoding

- Preserves characters, feasible to view ASCII subset (try an ad-hoc) 



New functions - most are intrinsic

- Non-Unicode input/output are "byte stream"
- Except where noted, functions (methods, BTW) throw `CharacterTranslationException` class 
 - Reason (enumeration): `InvalidEncoding`;
`InvalidCharacterReference`;
`UntranslatableCharacter`; (and `ToString`)
 - `BytePosition`
 - `CharacterPosition`
 - Description - "guts" of request cancel msg if exception not caught
 - `HexValue` - bad character/bytes




Sirius Software, Inc.

New functions - mostly intrinsic (*continued*)

- `%ebcdicString = unicode:UnicodeToEbcdic`
(`[CharacterEncode=bool]`)
 - Implicitly used for Unicode to EBCDIC conversion
 - `CharacterEncode=True` avoids exception (also note '&' for '&' in input)
- `%ebcdicString = asciiString:AsciiToEbcdic`
(`[CharacterEncode=bool]`)
 - `CharacterEncode=True` avoids exception (also note '&' for '&' in input)





New functions - mostly intrinsic (*continued*)

- `%unicode = string:EbcdicToUnicode`
(`[CharacterDecode=bool]`)
 - Implicitly used for EBCDIC to Unicode conversion
 - With `CharacterDecode=True`, good to create Unicode from mixed EBCDIC chars and `&#xhhhh`; Unicode chars 
- `%unicode = string:EbcdicToAscii`
(`[CharacterDecode=bool]`)
- `%unicode = utf8Stream:Utf8ToUnicode`
- `%utf8String = unicode:UnicodeToUtf8`
 - Doesn't throw `CharacterTranslationException`



New functions - mostly intrinsic (*continued*)

- %unicode = utf16Stream:Utf16ToUnicode
 - Good to create Unicode from "pure" U+hhhh literal 
- %utf16Stream = unicode:UnicodeToUtf16
 - Doesn't throw CharacterTranslationException
- %hexString = unicode:UnicodeToUtf16Hex
 - Doesn't throw CharacterTranslationException
 - Easy to display Unicode char so you can lookup in Unicode book or CD 
- %hexString = unicode:UnicodeToUtf8Hex
 - Doesn't throw CharacterTranslationException



Sirius Software, Inc.

New functions - mostly intrinsic (*continued*)

- %pos = unicode:UnicodeUntranslatablePosition
 - Doesn't throw CharacterTranslationException



New functions - mostly intrinsic (*continued*)

- %ebcdic =
ebcdicStream:EbcdicRemoveNonUnicode
 - Doesn't throw CharacterTranslationException
- %ebcdic =
ebcdicStream:EbcdicTranslateNonUnicode(char)
 - Doesn't throw CharacterTranslationException
- These are how you can deal with EBCDIC points that don't map to Unicode characters



New functions - mostly intrinsic (*continued*)

- %unicode = unicode:UnicodeReplace
 - Uses system-wide Unicode replacement table
 - Returns input Unicode string with chars from table replaced by corresponding strings
 - Default provided by Sirius: no replacements
 - See XML considerations
 - Best used with caution, if at all
 - Concrete example of what we had in mind:
 - ▶ e.g., replace UC=X'2122' with STR='(TM)'
 - Doesn't throw CharacterTranslationException



Sirius Software, Inc.

New functions - mostly intrinsic (*continued*)

- %replaceList = replaceList:AddUnicodeReplace
 - Update system-wide Unicode replacement table; format same as "permanent" assembler macro (with customized translation between Unicode and EBCDIC) - has met objections)-:
 - Doesn't throw CharacterTranslationException (but invalid input cancels request)
 - Input is list of pairs, each is Unicode code point and string to replace that point
 - Not intrinsic, BTW



XML API (not yet coded)

- Strings (i.e., values, names, prefixes, URIs) in XmlDocument stored in Unicode
- Most arguments and many results are Unicode
- Deserialization (e.g., LoadXml) options to let your application choose how and when to avoid character translation exceptions



Sirius Software, Inc.

XmlDoc strings stored as Unicode

- Representation issue is irrelevant to User Language programs
- Biggest change is that you now can choose to store full Unicode character set in XmlDoc
 - Default is to only allow Unicode characters which are translatable to EBCDIC



Sirius Software, Inc.

XML API Unicode arguments & results

- Under the covers, Unicode is used for any argument or result that stores or accesses string stored in XmlDocument, e.g.:
 - Value property: set or get
 - QName, Prefix, and URI function result
 - AddElement: name and value
 - XPath argument
 - SelectionNamespace property: set, get, and prefix argument



XML API Unicode arguments & results (*continued*)

- Existing code will run just as it did before, using implicit conversion between Unicode and EBCDIC
 - Previous API did not allow Unicode which wasn't translatable to EBCDIC (i.e., it did EBCDIC translation during deserialization)
 - ▶ Still that is the default
 - As long as applications only stored character (EBCDIC) data in XmlDocument



Deserialization control of untranslatable Unicode

- That's where the action is
- Default is to prevent Unicode characters which do not translate to EBCDIC
- Your application can choose to allow untranslatable Unicode
 - and maybe deal with it downstream
- You can also apply system-wide Unicode replacement table during deserialization
 - Or use replacement downstream
 - We're conservative about recommending



Deserialization control of untranslatable Unicode *(continued)*

- If your application allows untranslatable characters and there are any referenced "downstream" in the application, you can use the standalone facilities for Unicode to EBCDIC translation
- You can always add any Unicode to the XmlDocument, by providing argument of Unicode type
 - "Protection" of deserialization doesn't apply to this
 - But you need to work to create untranslatable Unicode characters



Sirius Software, Inc.

Deserialization control of untranslatable Unicode (*continued*)

- If LoadXml input is EBCDIC, it's converted to Unicode, throwing same exception cases as EbcDicToUnicode.
- Otherwise (UTF-8/16 or ASCII stream, or Unicode string) you can determine whether to assure or assist translatability to EBCDIC when XmlDocument is accessed: AllowUntranslatable and Replace args.
- These args also for other deserialization (WebReceive and HttpResponseMessage's ParseXml).



Sirius Software, Inc.

Deserialization control of untranslatable Unicode (*continued*)

- AllowUntranslatable: if False, exception is thrown if input contains any character which is not translatable to EBCDIC; default is False, assuring that parsed strings are translatable. If True, then application deals with possible untranslatability "downstream".
- Replace: if Standard, replacement of characters during parsing is performed using the system-wide Unicode replacement table (if you have one)



Deserialization control of untranslatable Unicode *(continued)*

- Control characters (such as X'7F' - DELEte) must be escaped with a character reference in the serialization of an XML document
 - Update to XmlDocument with control characters is allowed
- Null characters, X'00', are not allowed in XML documents
 - Neither deserialization into nor update



Assure translatability, no replacement

```
Try
    %doc:LoadXml(%serial)
Catch CharacterTranslationException
    %errors:add('Deserialization error')
End Try
```



Assure translatability, no replacement *(continued)*

- Default options of deserialization
- Assured that downstream (EBCDIC) access of parsed strings is translatable
 - Unless XmlDocument subsequently updated with Unicode input
- If omit Try/Catch, untranslatable character causes request cancellation



Assure translatability with replacement

Try

```
%doc:LoadXml(%serial, Replace=Standard)
```

Catch CharacterTranslationException

```
%errors:add('Deserialization error')
```

End Try



Sirius Software, Inc.

Assure translatability with replacement *(continued)*

- Assured that downstream (EBCDIC) downstream access of parsed strings is translatable
 - Unless XmlDocument subsequently updated with Unicode input
- If omit Try/Catch, untranslatable character causes request cancellation
- Again, don't strongly recommend using Replace



Allow untranslatability, no replacement

```
%doc:LoadXml(%serial, -  
AllowUntranslatable=True)
```



Sirius Software, Inc.

Allow untranslatability, no replacement *(continued)*

- Downstream (EBCDIC) access of parsed strings may cause `CharacterTranslationException`
 - If it does, employ explicit `UnicodeToEbcDic(CharacterDecode=True)` or possibly `UnicodeReplace`
- Since deserialization allows untranslatable characters, don't need `Try/Catch` for that case



Allow untranslatability with replacement

```
%doc:LoadXml(%serial, Replace=Standard, -  
AllowUntranslatable=True)
```



Sirius Software, Inc.

Allow untranslatability with replacement (*continued*)

- Downstream (EBCDIC) access of parsed strings may cause `CharacterTranslationException`
 - However, system-wide replacement table is used to convert any specified characters to translatable characters
 - Downstream access can employ explicit `UnicodeToEbcDic(CharacterDecode=True)`
 - `UnicodeReplace` would be redundant for parsed strings
- Since deserialization allows untranslatable characters, don't need Try/Catch for that case



Sirius Software, Inc.

Allow untranslatability with replacement *(continued)*

- To repeat, don't necessarily recommend UnicodeReplace



Sirius Software, Inc.

XmlDoc with untranslatable XmlDoc strings

■ Cases:

- Application may never access untranslatable characters
 - ▶ So don't need to deal with them
- Untranslatable characters may be accessed only for network serialization
 - ▶ Don't need to deal with them; UTF-8/16 conversion is correct and automatic



Sirius Software, Inc.

XmlDoc with untranslatable XmlDocument strings (continued)

■ Cases (continued):

- Untranslatable characters need to be converted to EBCDIC by application, and XmlDocument location of all untranslatable characters known in advance
 - ▶ Use explicit `UnicodeToEbcDic(CharacterEncode=True)`
 - ▶ Or possibly use `UnicodeReplace`
 - ▶ Only need it on nodes which may have untranslatable characters



Sirius Software, Inc.

XmlDoc with untranslatable XmlDocument strings (*continued*)

```
%ebc = %node:Value(eltWithNonAscii) -
```

```
  :UnicodeToEbcDic(CharacterEncode=True)
```

* (also may use UnicodeReplace, if you choose)

* If later converted to Unicode, be sure to use:

```
%unic = %ebc:EbcDicToUnicode(CharacterDecode=True)
```



Sirius Software, Inc.

XML & Unicode without new XML API

Prior to implementing Unicode XmlDocs:

```
If %doc:LoadXml(%str, 'ErrRet') Then
  %repstr = %str:Utf8ToUnicode:UnicodeReplace
  * Not what you want to do:
  * %repStr = -
%repStr:UnicodeToEbcDic(CharacterEncode=True)
  * Can you see why not?
  %errPos = %doc:LoadXml(%repStr, 'ErrRet')
  If %errPos Then
    ... log some error handling
  End If
End If
```



Sirius Software, Inc.

Edge case release boundary issues - all in XML API

- InvalidCharacter property deprecated or obsolesced
 - XML 1.1 approach - any control character can be stored - except Null
 - ▶ But don't allow Add/Insert/Value of untranslatable EBCDIC code point
 - Also allow character reference to control character when deserializing
 - Also probably deprecate InvalidCharacterPosition & IsValidString



Edge case release boundary issues - all in XML API (*continued*)

- Control characters now serialized using character references
- 7 questionable EBCDIC \leftrightarrow ASCII/UTF translations changed
- Noninvertible translations (2 EBCDIC sq bracket pairs) mean `%x:Value = '['`; `print %x:Value` may produce different sq bracket; change trans table?
- No more XPathOrder property
- Increased CCATEMP usage for Unicode strings
 - Probably very minor - maybe 10%?



Sirius Software, Inc.