Unicode Implementation Issues[*]

Shmuel (Seymour J.) Metz

August 2, 2023

Abstract

Classic Rexx supports character code pages in which every character can be represented in a single octet, and treats only ASCII letters as alphabetic. Unicode is nominally a 32-bit character set, with code points restricted to 17 16-bit planes, i.e., code points U+00 through U+10FFFF. This document describes basic issues and solutions for extending Rexx to support Unicode.

1 Background

Many modern languages support Unicode encoded as UTF-8 ([RFC 3629, STD 63]), and the IETF, in [RFC 5198], has mandated the use of UTF-8 with NFC in new protocols. Rexx will need to support UTF-8 if it is not to become a backwater, and that support should not break existing code.

2 Scope

This document only describes issues related to expanded alphabets and multi-octet representations of characters; it does not address such issues as bidirectional text. It presents suggested solutions for requirements that are presented in other documents. Also, most of the details are under active discussion, so everything here is provisional.

3 Nomenclature

The definitions given in [Glossary of Unicode Terms] and [Unicode] take precedence over those given here. Quoted text is taken from those sources. Except for definitions taken from official IETF and Unicode documents, the nomenclature used here is illustrative rather than normative; language design teams will formally define, e.g., method names, encoding of parameters. Most of the sections assume that there will be destinct string types for extended grapheme clusters, Unicode code points, legacy code pages and raw octets. However, there has been discussion of including metadata, in which case come of the classes might be merged.

.Legacy	The string class for strings in legacy code pages.
.octets	The string class for uninterprted octet strings.
.Ucp	The string class for strings of Unicode code points.
.Ugc	The string class for strings of Unicod extended grapheme clusters.
´	U+0301, COMBINING ACUTE ACCENT Set in bold to distinguish it from U+27, APOSTROPHE.
Bidi	”Abbreviation of bidirectional, in reference to mixed left-to-right and right-to-left text.”
BIF	Built In Function
BIM	Built In Method

[*] This is a working document for the Rexx Language Association (RexxLA). There will be one or more separate requirements documents. It is written in LaTeX 2022-11-01 and was rendered August 2, 2023 using luahbtex 1.17.0 (lualatex).

Block	”A grouping of characters within the Unicode encoding space used for organizing code charts. Each block is a uniquely named, continuous, non-overlapping range of code points, containing a multiple of 16 code points, and starting at a location that is a multiple of 16. A block may contain unassigned code points, which are reserved.”
BMP	Basic Multilingual Plane The first 64 Ki code points of Unicode, from U+0000 to U+FFFF.
BOM	Byte Order Mark: U+FEFF, ZERO WIDTH NON-BREAKING SPACE (ZWNBSP) Used as the first character to indicate byte order for UCS-2, UTF-16 and UCS-4; Optional as the first character for UTF-8.
É	U+C9, LATIN CAPITAL LETTER E WITH ACUTE Set in normal weight: U+45 U+0301 will be shown as ”É”.
é	U+E9, LATIN SMALL LETTER E WITH ACUTE Set in normal weight: U+65 U+0301 will be shown as ”é”.
encoded character	The smallest constituant of a Unicode string. ”The Unicode Standard does not define what is and is not a text element in different processes; instead, it defines elements called encoded characters. An encoded character is represented by a number from 0 to 10FFFF$_{16}$ called a code point.”
EGC	Extended grapheme cluster
GCGID	”Acronym for Graphic Character Global Identifier. These are listed in the IBM document Character Data Representation Architecture, Level 1, Registry SC09-1391.” See `https://www.ibm.com/downloads/cas/G01BQVRV`.
grapheme cluster	”A grapheme cluster consists of a base character followed by any number of continuing characters, where acontinuing character may include any nonspacing combining mark, certain spacing combining marks, or a join control.” [Unicode® Standard Annex 29] defines two types of grapheme clusters; ”An extended grapheme cluster is the same as a legacy grapheme cluster, with the addition of some other characters. The continuing characters are extended to include all spacing combining marks, such as the spacing (but dependent) vowel signs in Indic scripts.”
high surrogate	A code point in the range U+D800-U+DBFF, used as the first half of a surrogate pair.
IETF	Internet Engineering Task Force
introducer	The first octet in the UTF-8 encoding of a Unicode character beyond U+7F
low surrogate	A code point in the range U+DC00-U+DFFF, used as the second half of a surrogate pair.
NFC	Unicode ”Normalization Form C (NFC). A normalization form that erases any canonical differences, and generally produces a composed result. For example, a + umlaut is converted to ä in this form. This form most closely matches legacy usage. The formal definition is D120 in Section 3.11, Normalization Forms.” the normalization endorsed by the IETF
NFD	Unicode ”Normalization Form D (NFD). A normalization form that erases any canonical differences, and produces a decomposed result. For example, ä is converted to a + umlaut in this form. This form is most often used in internal processing, such as in collation. The formal definition is D118 in Section 3.11, Normalization Forms.”
NFKC	Unicode ”Normalization Form KC (NFKC). A normalization form that erases both canonical and compatibility differences, and generally produces a composed result: for example, the single character is converted to d + ž in this form. This form is commonly used in matching. The formal definition is D121 in Section 3.11, Normalization Forms.”

NFKD Unicode ”Normalization Form KD (NFKD). A normalization form that erases both canonical and compatibility differences, and produces a decomposed result: for example, the single character is converted to d + z + caron in this form. The formal definition is D119 in Section 3.11, Normalization Forms.”

octet 8-bit byte

Plane ”A range of 65,536 (1000016) contiguous Unicode code points, where the first code point is an integer multipleof 65,536 (1000016). Planes are numbered from 0 to 16, with the number being the first code point of the plane divided by 65,536. Thus Plane 0 is U+0000..U+FFFF, Plane 1 is U+10000..U+1FFFF, ..., and Plane 16 (1016) is U+100000..10FFFF. (Note that ISO/IEC 10646 uses hexadecimal notation for the plane numbers-for example, Plane B instead of Plane 11). (See Basic Multilingual Plane and supplementary planes.)”

RFC Request For Comments
A formal document published by the IETF defining, e.g., a protocol. RFC documents contain technical specifications and organizational notes for the Internet.

- Best Current Practice (BCP)
- Experimental
- Informational
- Proposed Standard
- Internet Standard (STD)
- Historic

surrogate A code point in the range U+D800-U+DFFF used to encode 21-bit code points into pairs of 16-bit bytes.

surrogate pair A high surrogate (in the range U+D800-U+DBFF) followed by a low surrogate (in the range U+DC00-U+DFFF), collectively representing a 21-bit code point.

TBD To Be Determined.

UCS Universal Character Set, ISO 10646, roughly equivalent to Unicode

UCS-2 A 16 bit subset of Unicode, containing only the BMP.

The Unicode Consortium A non-profit corporation devoted to developing, maintaining, and promoting software internationalization standards and data.

UTF-8 UCS Transformation Format 8 ([RFC 3629, STD 63]).
The encoding of Unicode endorsed by the IETF

UTF-8 octet sequence The sequence of octets representing a single Unicode code point. It may consist of a single ASCII character padded on the left with a zero bit, or of a one octet introducers followed by a 1-3 octet tail.

Code points	UTF-8 octet sequence
U+0000 - U+007F	0xxxxxxx
U+0080 - U+07FF	110xxxxx 10xxxxxx
U+0800 - U+FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+10000 - U+10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

UTF-16 UCS Transformation Format 16 ([RFC 2781]).

Some code samples use ooRexx notation. Those samples are illustrative rather than normative.

4 Statement of problem

4.1 Primary Problem

In Rexx, everything is a string; there are no classes, declarations or types.

While the ANSI standard does not mandate any particular character width for **Config_C2B()**, most if not all implementations use a width of 8, and the length of **c2x(***foo***)** is twice the length of *foo*.

A large body of existing code operates on binary data from external files, or accessed from memory via the **storage()** BIF, under the assumption that, e.g., **c2x()**, left(), **right()**, **substr()**, operate on octets. A UTF-8 introducer is treated the same as any other value.

A large body of existing code operates on text under the assumption that those facilities operate on characters.

The two categories overlap.

There is no conflict as long as each character is contained within a single octet. However, Unicode has code points beyond U+FF, and UTF-8 encoding of non-ASCII Unicode characters will require more than a single octet even for code points less than U+100 if they are beyond U+7F.

For example, the Unicode string 'Café' has 4 code points, 4 grapheme clusters and 5 octets in UTF-8 encoding while 'Cafe´' has 5 code points, 4 grapheme clusters and 6 octets in UTF-8 encoding, yet many text processing applications need to treat them as equivalent.

4.2 Secondary Problems

Rexx has built-in case conversion, but it is based on ASCII and can't even convert the accented letters found in, e.g., ISO-8859-1 (Latin-1), ISO-8859-15 (Latin-9), much less all those found in Unicode.

```
    foo = 'René'
    parse upper var foo bar
```

sets bar to RENé rather than to RENÉ.

In current implementations the range of characters is extremely small, and thus the **xrange()** only returns short strings. With Unicode the range expands to U+00 through U+10FFFF. That is far too large to allow as either an explict or implicit range in **xrange()**.

The current definition restricts symbols to ASCII characters, which excludes letters used in many languages. If the rules are extended to include Unicode letters and digits beyond ASCII, the rules for equality of symbols must be addressed. Are composed and decomposed strings identical? Are base characters identical to their compatibility alterrnates? Are subscript and superscript digits distinct from their ASCII counterparts?

When dealing with Unicode, different code sequences may have identical rendering due to the existence of both fully composed and combining code points and the exixtence of compatibility code points. Rexx needs a way to test two Unicode strings for equivalence.

There has been some discussion claiming a need to continue supporting legacy code pages.

There has been some discussion of the need to detect unassigned code points.

5 Solutions

Most of this section assumes that there will be three distict string types, and that some methods will not exist in all three types, or will behave differently. The type names used here are placeholders, and will be replaced once there is consensus on what to call them.

5.1 OPTIONS statement or directive

Define an option on the OPTIONS statement, or on a similar directive, to specify either raw octets or Unicode characters. This breaks programs that operate on both binary data and text.

OPTIONS mighy also specify a source encoding parameter, overriding any code page in an environment variable or file metadata.

5.2 Types

Allow Rexx variables to contain three distinct types of data: raw octets, Unicode code points and text, Define **storage()** as returning raw octets, and provide conversion functions ("casts"). There is an ongoing discussion as to whether all three are needed.

There should be no implicit conversion between **.octets** and Unicode strings.

There should be implicit conversion between Unicode types, but it need not be reversible, i.e., a conversion from .Ucp to .Ugc and back need not produce the same code points as the original

If support for legacy code pages is needed, a fourth string type could be defined. Strings of this type could include a code page attribute.

5.3 Width parameter

Add a width parameter to **c2x()** for Unicode code point string (raw octet strings may require input and output widths for UCS-2 AND UTF-16 data, and the utility of **c2x()** for grapheme clusters needs more analysis), and raise conditions with distinct error codes if any code point is out of range or if an invalid UTF-8 sequence is detected. A case could be made for using either the bit size or the digit size as the width.

Assuming that the width is in in terms of octets

```
foo = 'René'
bar = foo~UTF-8
/* assumes that width parameter is in octets */
say foo~c2x(2) /* Displays 00520065006E00E9  */
say bar~c2x    /* Displays 52656EC3A9        */
```

There is an ongoing discussion as to whether an when to allow implicit coercions of type in, e.g.,

```
foo = .octets~new
foo = 'René'
...
parse var foo ASCII 'é' .
say ASCII                    /* Displays Ren */
```

If there is no raw octet string added to the language, then **c2x()** might behave differently depending on whether an explicit width is provided.

5.4 Constants

Other languages allow specifying Unicode characters using either the hexadecimal value of the code point or the assigned name of the code point, e.g., U+E9 might be coded as \u{E9} or \u[LATIN SMALL LETTER E WITH ACUTE]. The syntax used for such constants should be consisten across Rexx variants, including rules for optional spaces between (hex) digits, and should take into account the recommendations in [RFC 5137, BCP 137]. Implementations should use the machine readable data bases published by the Unicode Consortium in order to ease migration to new versions of Unicode.

The form 'U+digits U+digits ...'U is clearer, but may break code that abuts the variable U with a string literal. The form 'U+digits U+digits ...'X is acceptable. There should be a discussion of syntax for named Unicode constants, e.g., '[COMBINING ACUTE ACCENT]'U is equivalent to 'U+0301'U. There has been some discussion of using the form '...'T and of escape conventions used in other languages.

Binary and hexadecimal literals are of type **.octets** and cannot be implicitly coerced to Unicode strings. Other string literals are legacy or Unicode and cannot be implictly coerced to **.octets**.

An alternative is to add a new notation using the ASCII ` as a framing character. This has the potential issue that it may be difficult to visually distinguish ` from '.

Another alternative is to use literals of the forms '...'type:codepage, '...'U:codepoints and '...'U:clusters. However, that makes it incompatinle with ooRexx.

The forms '...'{modifier} and '...'type{modifier} have no obvious conflict with existing syntax and has no abutment issues.

The following table is intended as a discussion point and should be updated whenever a consencuss is reached.

Example	Type	Semantics
'René'	Ucp Ugc	Unicode text by code point Unicode text by grapheme cluster (TBD)
'U+00E9'R	Ucp	Unicode text by code point
'U+00E9'Ucp	Ucp	Unicode text by code point
'U+00E9'Ugc	Ucp	Unicode text by grapheme cluster
'René'{ucp}	Ucp	Unicode text by code point
'René'{ugc}	Ucp	Unicode text by grapheme cluster
'René'{ISO8859-1}	Legacy	ISO 8859-1 legacy text

5.5 Error detection

The condition names given below are illustrative and not normative. Unicode BIF/BIMs should detect invalid input and signal the following conditions, with unique error codes.

INVALIDCODEPOINT	But carefully read 3.2 Conformance Requirements, Code Points Unassigned to Abstract Characters, p. 77, in [Unicode].
INVALIDUTF	An invalid UTF-8 octet sequence or invalid use of a UTF-16 surrogate.
NOENCODING	The operation requested requires a code page name.
NOTEXT	An operation was requested that is not valid for a binary file.
RANGE	A code point or other numeric entioty excees the permitted range. This might occur, e.g., when an application requuires that characters be limited to the BMP.

5.6 I/O

The **command** method of the stream classes should support a **CODEPAGE** option; attempting to set a codepage for a binary file should raise a **NOTTEXT** condition with a distinct error code. There should be an option or suboption controling whether to use Unicode if an apparent BOM is detected.

The default should probably be UTF-8 or ISO 8859-1 with switching to Unicode if a BOM is detected.

A discussion is needed on whether and when to create or discard byte order marks.

5.7 Case folding

The rules for case folding should take into account mathematical usage.

- U+1D400 through U+1D7FF are semantically distinct from other letters.
- Superscripts and subscripts have semantic significance.

5.8 Raw octet strings

The methods should include those of the ooRexx .string class, except that

- The unit of operation is the uninterpreted octet.
- The **makeString** method requires an encoding parameter.
- Parameters are raw octet strings.
- There are no caseless methods.

5.9 Legacy strings

If there is a .Legacy string type then the methods should have the same semantics as the existing methods for the .string class. If the new standard has a code page attribute then there should be an access method for it and the init or new method should allow specifying the code page.

The following Additional methods should be defined:

iconv	Convert from one code page to another and return a legacy string.
makeCodePageString	Return a Unicode string in which individual code points can be accessed
makeGraphemClusterString	Return a Unicode string in which only complete grapheme clusters can be accessed

5.10 Unicode strings

There should be subtypes depending on whether the unit of operation is the Unicode code point or the grapheme cluster. The methods should be those of the ooRexx .string class except:

- The caseless comparisons will use the [Unicode Character Database]. There will be variants to preserve or remove accents[1].
- There should be no **bit...** methods.
- Add an optional third parameter to the **upper** and **lower** methods to control whether to translate mon-ASCII characters and whether to strip accents.
- The following Additional methods should be defined:

makeCodePageString	Return a Unicode string in which individual code points can be accessed
makeGraphemClusterString	Return a Unicode string in which only complete grapheme clusters can be accessed
makeLegacyString	Return a legacy string for the specified code page in which individual octets can be accessed
makeOctets	Return a raw octet string using a specified encoding.[2]
NFC	Return a string normalized with Normalization Form C
NFD	Return a string normalized with Normalization Form D

There should be a description of how to handle conversion from Unicode code points and grapheme clusters that do not exist in the target code page.

- Replace with U+1A (SUB) or other specified character
- Raise SYNTAX with a unique error code
- Raise a new condition name with a unique error code

5.11 Methods

The following table summarizes some of the methods that differ among string types.

[1]Is Enye (ñ) considered an accented letter?
[2]default to UTF-8 or raise NOENCODING if no encoding specified?

Method	**.Legacy**	**.octets**	**.Ucp**	**.Ugc**
[]	by octet	by octet	by code point	by cluster
&	by octet	**n/a**	by code point	by cluster
|	by octet	**n/a**	by code point	by cluster
&&	by octet	**n/a**	by code point	by cluster
=	by octet	by octet	by code point	by cluster
¬=	by octet	by octet	by code point	by cluster
><	by octet	by octet	by code point	by cluster
<>	by octet	by octet	by code point	by cluster
<	by octet	by octet	by code point	by cluster
<=	by octet	by octet	by code point	by cluster
>	by octet	by octet	by code point	by cluster
¬>	by octet	by octet	by code point	by cluster
abbrev	by octet	by octet	by code point	by cluster
bitand	**n/a**	by octet	**n/a**	**n/a**
bitor	**n/a**	by octet	**n/a**	**n/a**
bitxor	**n/a**	by octet	**n/a**	**n/a**
c2b	8 bits	default 8 bits	Explicit width	**n/a**
c2x	8 bits	default 8 bits	Explicit width	**n/a**
center	by octet	**n/a**	by code point	by cluster
change	**.Legacy**	**.octets**	by code point	by cluster
find	by octet	by octet	by code point	by cluster
index	by octet	by octet	by code point	by cluster
left	by octet	by octet	by code point	by cluster
makeArray	by octet	by octet	by code point	by cluster
makeString	by octet	by octet	by code point	by cluster
pos	by octet	by octet	by code point	by cluster
right	by octet	by octet	by code point	by cluster
strip	by octet	by octet	by code point	by cluster
substr	by octet	by octet	by code point	by cluster
verify	by octet	by octet	by code point	by cluster

The backslash ("\") may be used in place of the Logical Not ("¬").

In addition, the makeString and makeString methods should allow caseless option parameters to control the class and attributes of the returned array elements or string, including:

Clusters	Unicode text with extended grapeme clusters as the abstract units.
Codepoints	Unicode text with Unicode scalars as the abstract units.
cp=legacy code page	Legacy or octet string with specified encoding.
cp=UTF-8	Legacy or octet string with UTF-8 encoding; functions like center and left will give unexpected results.
Legacy	Legacy string with default encoding unless cp= is also specified.
raw	Raw octet string; cp= must be specified.

5.12 Coercions

In addition to explicit casts via BIF/BIM, the following promotions from code-point strings to grapheme-cluster strings will be automatic

- Concatenation of code-point strings with grapheme-cluster strings. will be grapheme-cluster strings.
- Code point search arguments for grapheme-cluster strings. will be coerced to grapheme-cluster strings.
- Legacy strings tagged to a code page, in a context that requires Unicode, will be converted to the appropriate Unicode string type.

There will be no automatic promotion for octet strings, Unicode to legacy nor for legacy strings without code page tagging.

6 Bibliography

References

[Glossary of Unicode Terms] *Glossary of Unicode Terms*

[ISO/IEC 8859-1:1997 (E)] *Final Text of DIS 8859-1, 8-bit single-byte coded graphic character sets-Part 1: Latin alphabet No.1*

[ISO/IEC 10646:2020] *Information technology - Universal coded character set (UCS)*

[RFC 2781] *UTF-16, a transformation format of ISO 10646*

[RFC 3629, STD 63] *UTF-8, a transformation format of ISO 10646*

[RFC 5137, BCP 137] *ASCII Escaping of Unicode Characters*

[RFC 5198] *Unicode Format for Network Interchange*

[Unicode] *The Unicode Standard*

[Unicode Character Database] *Unicode Character Database (UCD)*

[Unicode CLDR Project] *Unicode Common Locale Data Repository (CLDR)*

[Unicode® Standard Annex 29] *Unicode® Standard Annex 29*
Unicode Text Segmentation

7 Resources

There are some useful tools available on the WWW:

Compart Unicode Look up, e.g., code point.