PostScript®
UNICODE EXTENSION REFERENCE

davidnewall.com

Introduction

This document contains detailed information about extensions to the PostScript language to support Unicode and UTF-8-encoded strings. They are designed to feel familiar to PostScript programmers, with a **ushow** extension corresponding to each **show** operator.

The layout of this document should also be familiar to PostScript programmers as it is similar to Adobe Systems' *PostScript Language Reference* (3rd edition), and the detailed descriptions of the extensions are mostly the same as the corresponding **show** operators.

A string is a sequence of 8-bit values, however, Unicode defines over one million code points and so **ushow** operators take an array of integers (Unicode code points or values) instead of a string.

Fonts in PostScript almost always contain thousands of glyphs, with up to 256 of them mapped via the **Encoding** array for use by the **show** operators. No corresponding map exists for Unicode, so the PostScript programmer must provide one along with the array of Unicode values when calling a **ushow** extension.

Few fonts, if any, contain glyphs for every Unicode code point and it would be highly wasteful of space to use an array for the Unicode map, even were it technically possible (PostScript implementations typically restrict arrays to 65,535 elements). For this reason, code points are mapped to glyphs using a dictionary, with the Unicode value as the *key* and the glyph name (or an array of names) as the *value*.

There is no fixed standard for glyph names in fonts. Although most fonts use the same names for ASCII glyphs, they often use different names for other glyphs. For this reason, a specific map needs to be created for each font used.

Fontforge (https://fontforge.org) can extract a list of a font's glyph names and corresponding Unicode values. It does this when saving a font in .otf or .ttf format with the "output glyph map" option enabled. This instructs Fontforge to create a .g2n file which can easily be converted to a Unicode map, for example with the following awk script:

```
BEGIN {print "<<"}
/GLYPHID .*PSNAME .*UNICODE/ {print "16#"$6"/"$4}
END {print ">> readonly"}
```

Adobe published a glyph list ("AGL") which they intended to be used to determine the Unicode value corresponding to various glyph names. It can be reversed to create an encoding map, but this is not recommended. It maps a mere 3,680 code points whereas tens of thousands are needed to cover just the common written languages. Further, many fonts use different names than AGL and include glyphs which are not listed in it. Using AGL may seem easy but is likely to result in characters not being painted even though they are present in the font.

Despite this recommendation, some programmers will use AGL, which complicates matters because it lists multiple names for some Unicode values. (This is why the map passed to unicodeshow can contain either a name or array of names for any Unicode value.)

This document is divided into two sections:

- Section 1 gives a summary of the extensions.
- Section 2 provides detailed descriptions of all extensions.

Each description is presented in the same format as the PostScript Language Reference:

extension *argument*$_1$ ... *argument*$_n$ **extension** *result*$_1$ ... *result*$_m$

A detailed explanation of the extension.

Example

An example of the use of this extension.
The symbol ⇒ designates the values left on the operand stack by the example.

Errors: **A list of errors that this extension might execute**
See Also: **A list of related extensions**

At the head of an extension description, *argument*$_1$ through *argument*$_n$ are the arguments that the extension requires, with *argument*$_n$ being the topmost element on the operand stack. The extension pops these objects from the operand stack and consumes them; then it executes. After executing, the extension leaves the objects *result*$_1$ through *result*$_m$ on the stack, with *result*$_m$ being the topmost element.

Normally, the operand and result names suggest either their types or their uses. The following table explains the names which are peculiar to these extensions:

NAME	DESCRIPTION
map	A dictionary which maps Unicode values to glyph names
unicode	An array of Unicode code points

SECTION 1

Unicode Extension Summary

string **utf8decode** *unicode*	decode UTF-8 string giving array of Unicode values
map unicode **ushow** -	paint glyphs for *unicode* in current font
a_x a_y *map unicode* **aushow** -	Add (a_x, a_y) to width of each glyph while showing *unicode*
c_x c_y *int map unicode* **widthushow** -	Add (c_x, c_y) *to width of glyph for int* while showing *unicode*
c_x c_y *int* a_x a_y *map unicode* **awidthushow** -	Combine effects of *aushow* and *widthushow*
map unicode numarray|numstring **xushow** -	Paint glyphs for *unicode* using *x* widths in *numarray|numstring*
map unicode numarray|numstring **xyushow** -	Paint glyphs for *unicode* using *x* and *y* widths in *numarray|numstring*
map unicode numarray|numstring **yushow** -	Paint glyphs for *unicode* using *y* widths in *numarray|numstring*
map unicode **ustringwidth** w_x w_y	Return width of glyphs for *unicode* in current font
proc unicode **kushow** -	Execute *proc* between glyphs shown from *unicode*

SECTION 2

Unicode Extension Details

aushow *a_x a_y map unicode* **aushow** -

Paints glyphs for the Unicode values in *unicode* in a manner similar to **ushow**; however, while doing so, **aushow** adjusts the width of each glyph shown by adding *a_x* to the glyph's *x* width and *a_y to its y* width, thus modifying the spacing between glyphs. The numbers *a_x and a_y are x* and *y* displacements in the user coordinate system, not in the glyph coordinate system.

This extension enables fitting a string of text to a specific width by adjusting all the spacing between glyphs by a uniform amount. For a discussion of glyph widths, see Section 5.4, "Glyph Metric Information" of *PostScript Language Reference.*

Example

```
AdobeGlyphList dup length dict begin {
  exch [ exch currentdict 3 index known
    {currentdict 3 index get aload pop}
  if ] def
} forall currentdict end /Map exch def
/Helvetica 8 selectfont
14 67 moveto
"NormalSpace"      Map [8220 78 111 114 109 97 108 83 112 97 99 101 8221] ushow
14 47 moveto 4 0
"WideSpace"        Map [8220 87 105 100 101 83 112 97 99 101 8221] aushow
```

Errors: **invalidfont, nocurrentpoint, typecheck**
See Also: **auwidthshow, kushow, ushow, widthushow, xushow, xyushow, yushow**

awidthushow *c_x c_y int a_x a_y map unicode* **awidthushow** -

Paints glyphs for the Unicode values in *unicode* in a manner similar to **ushow**, but combines the special effects of **aushow** and **widthushow**. **awidthushow** adjusts the width of each glyph shown by adding *a_x to its x* width and *a_y to its y* width, thus modifying the spacing between glyphs. Furthermore, **awidthushow** modifies the width of each occurrence of the glyph for the Unicode value *int* by an additional amount (c_x, c_y).

This extension enables fitting a string of text to a specific width by adjusting the spacing between all glyphs by a uniform amount, while independently controlling the width of the glyph for a specific character, such as the space. For a discussion of glyph widths, see Section 5.4, "Glyph Metric Information" of *PostScript Language Reference.*

Example

```
AdobeGlyphList dup length dict begin {
  exch [ exch currentdict 3 index known
    {currentdict 3 index get aload pop}
  if ] def
} forall currentdict end /Map exch def
/Helvetica 8 selectfont
14 67 moveto
"Normal Space"  Map [8220 78 111 114 109 97 108 32 83 112 97 99 101 8221] ushow
14 47 moveto
5 0 16#20 2 0
"W i d e  S p a c e"  Map [8220 87 105 100 101 32 83 112 97 99 101 8221] awidthushow
```

Errors: **invalidfont, nocurrentpoint, typecheck**
See Also: **aushow, kushow, ushow, widthushow, xushow, xyushow, yushow**

kushow *proc map unicode* **kushow** -

Paints glyphs for the Unicode values in *unicode* in a manner similar to **ushow**, but allows program intervention between characters. If the values in *unicode* are int_0, int_1, ... int_n, **kushow** proceeds as follows: First it shows the glyph for int_0 at the current point, updating the current point by the width of that glyph. Then it pushes int_0 and int_1 on the operand stack and executes *proc*. *proc* may perform any actions it wishes; typically, it will modify the current point to affect the subsequent placement of the glyph for int_1. **kushow** continues by showing the glyph for int_1, pushing int_1 and int_2 on the stack, executing *proc*, and so on. It finishes by pushing int_{n-1} and int_n on the stack, executing *proc*, and finally showing the glyph for int_n.

When proc is called for the first time, the graphics state (in particular, the current transformation matrix) is the same as it was at the time **kushow** was invoked, except that the current point has been updated by the width of the glyph for int_0. Execution of *proc* is permitted to have any side effects, including changes to the graphics state. Such changes persist from one call of *proc* to the next and may affect graphical output for the remainder of **kushow**'s execution and afterward. When *proc* completes execution, the value of currentfont is restored. The name **kushow** is derived from "kern-show." To kern glyphs is to adjust the spacing between adjacent glyphs in order to achieve a visually pleasing result. The **kushow** operator enables user-defined kerning and other manipulations, because arbitrary computations can be performed between pairs of glyphs.

kushow can be applied only to base fonts. An **invalidfont** error occurs if the current font is a composite font, a CIDFont, or defines neither CharProcs not CharStrings.

Errors: **invalidfont, nocurrentpoint, typecheck**
See Also: **aushow, awidthushow, kushow, ushow, widthushow, xushow, xyushow, yushow**

ushow *map unicode* **ushow** -

Paints glyphs for the Unicode values in *unicode* on the current page starting at the current point, using the font face, size, and orientation specified by the current font (as returned by **currentfont**).

map maps Unicode values to potential glyph names, either as a single name or an array of names. The current font's **CharStrings** dictionary is searched for each potential glyph name, and the first glyph found is selected. If the current font has no *CharStrings* dictionary, it's **CharProcs** dictionary is used instead. An **invalidfont** error occurs if the font has neither **CharStrings** nor **CharProcs**.

If a Unicode value is not mapped by *map*, or if none of the potential glyph names appears in the font's **CharStrings** dictionary, **uniXXXX** is used as a fallback name for values less than 16#10000 and **uXXXXXXXX** for values of 16#10000 and over, where XXXXXX is the value in hexadecimal.

The spacing from each glyph to the next is determined by the glyph's width, which is an (*x, y*) displacement that is part of the glyph description. When it is finished, **ushow** adjusts the current point in the graphics state by the sum of the widths of all the glyphs shown. **ushow** requires that the current point initially be defined (for example, by **moveto**); otherwise, a **nocurrentpoint** error occurs.

See Chapter 5 of *PostScript Language Reference* for complete information about the definition, manipulation, and rendition of fonts.

Errors: **invalidfont, nocurrentpoint, typecheck**
See Also: **moveto, setfont, aushow, auwidthshow, kushow, widthushow, xushow, xyushow, yushow**

ustringwidth *map unicode* **ustringwidth** w_x w_y

Calculates the change in the current point that would occur if *map* and *unicode* were given as operands to **ushow** with the current font. w_x and w_y are computed by adding together the width vectors of all the individual glyphs for *unicode* and converting the result to user space. They form a distance vector in the x and y dimensions describing the width of the glyphs for the entire string in user space. See Section 5.4, "Glyph Metric Information," of *PostScript Language Reference* for a discussion of glyph widths.

To obtain the glyph widths, **ustringwidth** executes the descriptions of one or more of the glyphs in the current font and may cause the results to be placed in the font cache. However, **ustringwidth** prevents the graphics operators that are executed from painting anything onto the current page.

Note that the width returned by **ustringwidth** is defined as movement of the current point. It has nothing to do with the dimensions of the glyph outlines (see **charpath** and **pathbbox**).

Errors: **invalidfont, typecheck**
See Also: **setfont, ushow**

utf8decode *string* **utf8decode** *unicode*

Returns an array object containing Unicode values decoded from *string*, which is encoded using UTF-8 variable-width character encoding. Invalid UTF-8 sequences are replaced with Unicode Character 'REPLACEMENT CHARACTER' (U+FFFD).

Example

```
(\342\200\234UTF-8\342\200\235) utf8decode  ⇒  [8220 85 84 70 45 56 8221]
```

widthushow c_x c_y *int map unicode* **widthushow** -

Paints glyphs for the Unicode values in *unicode* in a manner similar to **ushow**; however, while doing so, it adjusts the width of each occurrence of the glyph for Unicode value *int* shown by adding c_x to its *x* width and c_y *to its y* width, thus modifying the spacing between it and the next glyph. This operator enables fitting a string of text to a specific width by adjusting the width of the glyph for a specific character, such as the space character.

Example

```
AdobeGlyphList dup length dict begin {
  exch [ exch currentdict 3 index known
    {currentdict 3 index get aload pop}
  if ] def
} forall currentdict end /Map exch def
/Helvetica 8 selectfont
14 67 moveto
Map [8220 78 111 114 109 97 108 32 83 112 97 99 101 8221] ushow
14 47 moveto 6 0 16#20
Map [8220 87 105 100 101 32 83 112 97 99 101 8221] widthushow
```

"Normal Space"

"Wide Space"

Errors: **invalidfont, nocurrentpoint, typecheck**
See Also: **aushow, auwidthshow, kushow, ushow, xushow, xyushow, yushow, ustringwidth**

xushow *map unicode numarray* **xushow** -
xushow *map unicode numstring* **xushow** -

Is similar to **xyushow**; however, for each glyph shown, **xushow** extracts only one number from *numarray* or *numstring*. It uses that number as the *x* displacement and the value 0 as the y displacement. In all other respects, **xushow** behaves the same as **xyushow**.

Errors: **invalidfont, nocurrentpoint, typecheck**
See Also: **ushow, xyushow, yushow**

xyushow *map unicode numarray* **xyushow** -
xyushow *map unicode numstring* **xyushow** -

Paints glyphs for the Unicode values in *unicode* in a manner similar to **ushow**. After painting each glyph, it extracts two successive numbers from the array *numarray* or the encoded number string *numstring*. These two numbers, interpreted in user space, determine the position of the origin of the next glyph relative to the origin of the glyph just shown. The first number is the *x*

displacement and the second number is the *y* displacement. In other words, the two numbers override the glyph's normal width.

If *numarray* or *numstring* is exhausted before all values in *unicode* have been shown, a **rangecheck** error occurs. See Section 5.1.4, "Glyph Positioning," in *PostScript Language Reference* for information about **xyshow**, and Section 3.14.5, "Encoded Number Strings," for an explanation of the *numstring* operand.

Errors: **invalidfont, nocurrentpoint, rangecheck, typecheck**
See Also: **ushow, xushow, yushow**

yushow *map unicode numarray* **yushow** -
yushow *map unicode numstring* **yushow** -

Is similar to **xyushow**; however, for each glyph shown, **yushow** extracts only one number from *numarray* or *numstring*. It uses that number as the *y* displacement and the value 0 as the x displacement. In all other respects, **yushow** behaves the same as **xushow**.

Errors: **invalidfont, nocurrentpoint, rangecheck, typecheck**
See Also: **ushow, xyushow, yushow**