FONT CODE CHANGES

By: Ron Kaplan

This document last edited on July 11, 2025.

Introduction

This document describes changes made to the font implementation in order to simplify the code, remove some unecessary (and sometimes broken) features, and to generalize some of the interfaces. The new code also allows the chain of font coercions to fill in not just entirely missing character sets but missing characters in character sets that are only partially populated.

Font data structures

The legacy fields FBBOX FBOY BBDX BBDY \SFLKerns \SFRWidths of FONTDESCRIPTOR have been removed and new flag fields FONTCOMPLETEP and FONTHASLEFTKERNS have been added. FONTCOMPLETEP is T if all character sets of the font are ‘‘glyph-complete’’ as defined below. FONTHASLEFTKERNS is T if at least one character set has left-kern information (althoughthe interpretation of kerning information is still not well understood). The

Element 256 of the FONTCHARSETVECTOR has been reserved for a slug CHARSETINFO with a slug (black box or equivalent) entry for every character. A pointer to that single slug CHARSETINFO is installed in the vector for a character set that is requested but for which it is known that is has no real characters. Thus, an element of the vector is NIL if a request has not yet been made and is always a CHARSETINFO (maybe the slug) after the first request.

Flag fields CSSLUGP and CSCOMPLETEP and a pointer field CSINFOPROPS have been added to the CHARSETINFO data type. CSSLUGP is T for every slug CHARSETINFO. CSCOMPLETEP is T if the CHARSETINFO is glyph-complete (a slug charset is always glyph-complete). CSINFOPROPS holds an alist of device-independent or device-specific properties that may be associated with a charset.

(CHARSETPROP *CSINFO PROP NEWVALUE*) [Macro]

CSINFO is a CHARSETINFO. If *NEWVALUE* is provided, it is stored as the new *PROP* value of *CSINFO*, and NEWVALUE is returned. Otherwise, returns the current *PROP* value of *CSINFO*.

Font creation

The function FONTCREATE has been factored into two subfunctions, \FONT.CHECKARGS and FONTCREATE1. \FONT.CHECKARGS interprets the flexible ways in which the properties of the desired font can be expressed and/or defaulted, and checks them for validity. FONTCREATE1 finds or creates the font, knowing that the arguments have been standardized and validated.

(\FONT.CHECKARGS *FAMILY SIZE FACE ROTATION DEVICE CHARSET*) [Function]

The arguments are examined and possibly coerced or defaulted to values that would be sensible specifications for a requested font. Thus, *FAMILY* can be an atomic family name, but also a list of font parameters, or a fontdescriptor or font class from which the desired properties can be extracted. *SIZE* must reduce to a postive integer, *FACE* must be convertable to a (weight slope expansion) list, *DEVICE* defaults to DISPLAY, *CHARSET* defaults to 0, etc. If the arguments are coerceable, \FONT.CHECKARGS returns a multiple-value vector of the decoded values for *FAMILY SIZE FACE ROTATION DEVICE CHARSET FONTX*, where the last *FONTX* is either NIL or the known font descriptor that may have emerged from the interpretation of the other arguments.

\FONT.CHECKARGS causes an error if any of the arguments cannot be properly deciphered (e.g. *FAMILY* does not map to an atom, ■*FACE* is not a face, etc).

\FONT.CHECKARGS is called by FONTCREATE, but also by other functions that interpret font specifications.

(FONTCREATE1 *FAMILY SIZE FACE ROTATION DEVICE CHARSET*) [Function]

Given that the arguments have been validated, this finds or constructs the corresponding fontdescriptor and ensures that the character set *CHARSET* is instantiated and *character-complete*, as defined below. Returns NIL (without error) if the arguments do not select a font with at least one defined character set (usually character set 0).

If FONTCREATE1 returns NIL, FONTCREATE causes the FONT NOT FOUND error unless its NOERRORFLG argument is T. The error will specify the validated arguments as of the FONTCREATE, even if the failure was down some chain of coercions--the error pops to the top.

Note that the error behavior is different than previously implemented. Before, an error would be generated if the requested *CHARSET* is not defined even though there is information to instantiate at least one other character set in the font. Now it is judged that the *font* exists but the requested character set may be filled with slugs (black boxes). Asking for *FAMILY* FOO will cause an error, but asking for GACHA character-set 5 (which is undefined in MCCS) will not. Characters in that character set will display as black boxes.

FONTCREATE1 retrieves a specified fontdescriptor from the \FONTSINCORE cache, and adds to that cache if it creates a new descriptor. This is done by the PUTMULTI and GETMULTI macros of the MULTI-ALIST package. The previous function used for this purpose, \LOOKUPFONTSINCORE, has been removed.

(\READCHARSET *FAMILY SIZE FACE ROTATION DEVICE CHARSET)* [Function]

This is a new primitive function for creating the CHARSETINFO of a requested character set based on information from some external source, presumably a file. It generalizes the previous \READDISPLAYFONTFILE to multiple file formats for DISPLAY and eventually for different font devices.

This calls the new function FONTFILES to get an ordered list of candidate files for the specified font-charset, based on standard font-file naming conventions. For each file, it runs down the elements on a device-specific list of character-set functions provided as the top-level value of a variable [device]CHARSETFNS (e.g. DISPLAYCHARSETFNS). Each entry on this list is a triple of the form

(typename predicatefn getcharsetinfofn)

If the predicatefn applied to a filename is true, then the getcharsetinfofn is applied. If that produces a CHARSETINFO, that is the value returned; otherwise the next triple is considered. For some file types (e.g. STRIKE, AC) the character set is embedded in the filename. *CHARSET* is passed as a parameter for formats that can incorporate information about multiple character sets (e.g. Medley font-format files). The current entrties on DISPLAYCHARSETFNS are:

```
    ( ( AC    ACFONT.FILEP    ACFONT.GETCHARSET)
      ( MEDLEYFONT    MEDLEYFONT.FILEP    MEDLEYFONT.GETCHARSET)
      ( STRIKE    STRIKEFONT.FILEP    STRIKEFONT.GETCHARSET))
```

(ACFONT.FILEP and ACFONT.GETCHARSET are new functions on AFONT. The MEDLEYFONT functions are on MEDLEYFONTFORMAT. STRIKEFONT.FILEP is a new function on FONT and STRIKEFONT.GETCHARSET on FONT is a rename of the previous \READSTRIKEFONTFILE.)

\READCHARSET records the name of the file as the FILE property of the CHARSETINFO and it stores the list of charset parameters (*FAMILY SIZE* etc.) as the SOURCE property.

It also stores the character encoding of the charset as the value of the CSCHARENCODING property. The character encoding may be included in the CHARSETINFO returned by the getcharsetinfofn (as for MEDLEYFONT files). Otherwise for character sets other than 0 the character encoding defaults to MCCS, and for character set 0 the character encoding is determined from the family. This is XCCS$ for families in the list NSFONTFAMILIES (Classic, Modern...), ALTOTEXTFONT for families in the list ALTOFONTFAMILIES (Gacha, Helvetica...), and *FAMILY* itself for other families (Hippo, Math...).

\READCHARSET returns NIL if it cannot find/read an appropriate CHARSETINFO.

(FONTFILES *FAMILY SIZE FACE ROTATION DEVICE CHARSET DIRLST EXTLST)* [Function]

This new function returns a list of the standard-named files that contain information about *CHARSET* in the specified font. *DIRLST* and *EXTLST* are optional lists of the directories and extensions to be searched for. If *DIRLST* is not provided, then the value of the variable [DEVICE]FONTDIRECTORIES (e.g. DISPLAYFONTDIRECTORES for DEVICE=DISPLAY) is used. Similarly, if EXTLST is not specified the value of [DEVICE]FONTEXTENSIONS is used. The initial values of DISPLAYFONTDIRECTORIESand DISPLAYFONTEXTENSIONS are

```
    ( [MEDLEYDIR]/<fonts>medleydisplayfonts    [MEDLEYDIR]/fonts/displayfonts )
```
and
```
    ( MEDLEYDISPLAYFONT    DISPLAYFONT    STRIKE )
```
Eventually these will be reduced to just the Medley-formatt directory and extension.

FONTFILES considers as candidates the standard file names that begin with the *CHARSET* subdirectory and end with *CHARSET* (e.g. c357>GACHA10-MIR-C357) and also file names without any charset indication (simply GACHA10-MIR). The latter is appropriate for multi-charset Medley-font formatted files or fonts for devices that do not distribute their charsets into separate subdirectories.

Font existence and availability

(FONTEXISTS? *FAMILY SIZE FACE ROTATION DEVICE*) [Function]

This new function returns a non-NIL value if a font descriptor with the specified parameters can be retrieved from \FONTSINCORE or can be constructed from the information in a font-file. It records the result of the search in a separate cache, \FONTEXISTS?-CACHE, to avoid future lookups. This is not equivalent to a direct call to FONTSAVAILABLE because this takes into account the possibility of coercing the requested parameters to the parameters of some other font (e.g. Modern 60 "exists" because it coerces to Modern 24).

FONTEXISTS? [IMAGESTREAMTYPES method]

This new component of the IMAGESTREAMTYPES entry for a font device (DISPLAY, POSTSCRIPT, PDF, HTML...) is a function of the usual (pre-validated) font parameters. It returns the non-NIL value that will be stored in the \FONTEXISTS?-CACHE if it would be possible to create the specified font from file or coercion data.

A separate new function, \FONTSAVAILABLE.INCORE. is provided to produce the descriptions of the in-core fonts for FONTSAVAILABLE. This function interprets the * wild cards for font searches, replacing the deprecated \LOOKUPFONTSINCORE for the in-core search.

(\FONTSAVAILABLE.INCORE *FAMILY SIZE FACE ROTATION DEVICE*) [Function]

(FONTFILEP *FILE DEVICE*) [Function]

This returns the typename of an element of [DEVICE]CHARSETFNS whose predicatefn applied to *FILE* is non-NIL.

The function \SEARCHFONTFILES has been modified so that its *DIRLST* and *EXTLST* arguments default to the global values of variables derived from the *DEVICE*. It may not be necessary to maintain specialized versions (like the previous \SEARCHDISPLAYFONTFILES and \SEARCHINTERPRESSFONTFILES) just to pass those values.

(FLUSHFONTSINCORE *FAMILY SIZE FACE ROTATION DEVICE*) [Function]

Removes the specified fonts(s) from \FONTSINCORE. As for FONTSAVAILABLE, the wild-card * may appear for any of the parameters.

Charset coercion and glyph completion

Previous code implemented several strategies to simulate a requested character set or font when there is no explicit source data to match that request. There is no file for Greek characters in Gacha 10, say, but MISSINGCHARSETDISPLAYFONTCOERCIONS specified that the Greek character set of Terminal 10 may be a good approximation, and if that did not exit, then Modern was the heuristically next best place to look. If there was no source data for an italic or bold face, the previous code searched for characters in another face that could be algorithmically slanted or fattened. And the variable MISSINGDISPLAYFONTCOERCIONS defined how one whole font could be substituted for another, so that a request for Helvetica 2 would best be satisfied entirely from the Helvetica 4 source files and coercions.

The code for implementing these coercions was recursively tangled, to say the least. But the code was also flawed in another way: its coercions applied at the character-set level and not at the level of the glyphs for individual characters. Consider a request for character set 357 in Terminal 10. That request is immediately satisfied by the file TERMINAL10-MRR-C357.DISPLAYFONT, and on that basis the code would see no need for further coercion. But that particular file only has information about half a dozen characters, the other symbols in c357 will show up as black-box slugs. If that file had not been there, the coercion would have found the file MODERN10-C357.DISPLAYFONT, which instantiates virtually all the symbols in character set 357. In effect, the partial Terminal-specific file blocked the possibility of replacing its slugs with the glyphs for a much larger collection of characters.

The resulting charsetinfo is thus not “glyph-complete” in the sense that it does not incorporate all the information that is available directly or indirectly for the characters in that set. Similarly, the font descriptor that includes that charsetinfo is also not glyph-complete: it includes at least one charset that is not glyph-complete.

The new code maintains the same basic strategy of looking first for an appropriate character-set file and failing that, coercing for a character set in a different font. But the best charsetinfo, when it is finally located, is then further processed if it is not already marked as glyph-complete (its CSCOMPLETEP is NIL). The function COMPLETE.CHARSET is called to see if any of the slugs in that charsetinfo can be individually replaced by glyphs in charsets further down in the coercion chain.

(COMPLETE.CHARSET *CSINFO FAMILY SIZE FACE ROTATION DEVICE CHARSET COERCIONS FONTDESC*) [Function]

For each code between 0 to 255, if it is instantiated as a slug in *CSINFO*, COMPLETE.CHARSET searches through the coercions to see if it is not a slug in some later character set. If a non-slug entry exists for that code, its glyph information is copied up into *CSINFO.* At the end, *CSINFO* is marked as glyph-complete whether or not anything has been changed. But if at least one slug was replaced, the original SOURCE property of *CSINFO* is changed to reflect the fact that it has now been specialized to the *FAMILY SIZE* etc. parameters of this font. (If no replacements are made, the original source is a direct reference to the charset information, skipping over any intermediate coercion steps.)

The slugs in a display character set are detected by the function SLUGCHARP.DISPLAY.

(SLUGCHARP.DISPLAY *CODE FONT/CHARSETINFO*) [Function]

If the second argument is a font descriptor, *CODE* is interpreted as a character code in the full range 0 to 65535, and the target character set in that font is extracted with *CODE*’s high-order byte. Otherwise, *CODE* must be an integer between 0 and 255. The predicate is true if the bitmap for the code in this character set has the same offset as the bitmap of the slug (which is stored at position 256 in every charsetinfo). (It remains to generalize this predicate to POSTSCRIPT, HTML, and other font devices.)

(COMPLETE.FONT *FONT EVENIFCOMPLETE*) [Function]

Unless *FONT* is already marked as glyph-complete (or *EVENIFCOMPLETE* is T), this function makes sure that every charset from 0 to 255 that can exist directly or indirectly for *FONT* has been instantiated and is glyph-complete. At the end the FONTCOMPLETEP flag is set to T.

The variable name MISSINGCHARSETDISPLAYFONTCOERCIONS is not only ponderous but also puts the focus on the character sets instead of the individual glyphs. In the new code the name of this variable is changed to DISPLAYGLYPHCOERCIONS

DISPLAYGLYPHCOERCIONS [Variable]

And the variable MISSINGDISPLAYFONTCOERCIONS has also been renamed:

DISPLAYFONTCOERCIONS [Variable]

Analogous variables may be defined for other font devices if it turns out that this coercion logic is more generally useful.

Miscellaneous

The new public function MOVEFONTCHARS moves glyph information from one font to another.

(MOVEFONTCHARS *PAIRS DEVICE DESTFONT DEFAULTSOURCEFONT*) [Function]

PAIRS is a list of (source destination) pairs, where each source is either a character indicator (a character name or code), or a pair consisting of a character indicator and a font. If a source character name/code is not associated with a font, then *DEFAULTSOURCEFONT* will be used, otherwise *DESTFONT.* Each destination is a character name or code. The glyph information for each source will be moved from its font to the destination position in *DESTFONT*. This moves all of the character parameters from one place to another, including the character bitmap for display fonts. This function can be used to fill in glyphs arbitrarily from other fonts/sizes/faces as a backup to the more general glyph-coercion specifications.

(FLUSHFONTSINCORE *FAMILY SIZE FACE ROTATION DEVICE*) [Function]

Removes the indicated fontdescriptor(s) from the internal cache. A wildcard * can be provided as any of the arguments.

The function FONTPROP has been extended so that it returns for the property CHARENCODING the font's FONTCHARENCODING. and for the property CHARSETS a list of the numbers of the currently instantiated non-slug charsets. Other charsets may be available, just not yet requested and loaded into memory.

The function/macro \GETFONTDESC was a confusing synonym for \COERCEFONTDESC, and \COERCEFONTDESC had some unbounded recursive branches through FONTCREATE. \COERCEFONTDESC has been internalized as a subfunction of FONTCREATE, and FONTCREATE has been substituted for all other occurrences of both of these.

There is a new file internal/FONT-DEBUG with a random set of tools that have been helpful in debugging and testing these changes.

[Other changes to be documented as appropriate]