Medley Character Code Standard

By: Ron Kaplan

This document last edited on September 3, 2025.

Introduction

This document describes the Medley Character Code Standard (MCCS), the coding convention for characters. atoms, and strings internal to a Medley environment. It also introduces the :MCCS external forma, the tools for operating on MCCS-coded files and data, and the techniques for translating from other legacy coding conventions to MCCS. The external format and translation functions are defined in the source file MCCS. The file MCCS also includes the :XCCS external format that formerly resided on the file XCCS. MCCS replaces XCCS in the loadup.

The MCCS character encoding and :MCCS external format

The MCCS encoding is the same as the Xerox Character Code (XCCS) Standard but for the following differences shown in the first two columns of the following table:

	XCCS	MCCS	XCCS$
0,44 =x24	¤	$	$
0,244=xA4	$	¤	¤
0,137=x5F	_	←	_
0,254=xAC	←	_	←
0,136=x5E	^	↑	^
0,255=xAD	↑	^	↑

The MCCS encoding differs from true XCCS in that the dollar sign and currency are swapped (the first two lines above) and the arrows and caret/underline are also swapped). But to be clear, the character encoding of Medley's current :XCCS external format is not exactly XCCS. Rather, it is the minor variant shown in the third column wherein only the dollar and currency are swapped. This is the XCCS$ character encoding that our so-called NS display fonts (Classic, Modern, Terminal...) embody. The difference shows up for example, in the fact that printing to Interpress, a true XCCS device, requires a translation of the dollar sign code for the hardcopy file to have the same appearance as the display.

Along with the MCCS character encoding there is an :MCCS external format. The :MCCS external format piggy-backs on the Xerox NS representation of multi-byte characters as runs of characters in the same character set, reserving byte 255/FF as the character-set shifting character. But it differs from the :XCCS external format in that it explicitly assumes characters are represented in the MCCS encoding.

(Note: Unlike the UTF-8 representation of codes, the NS representation and thus the :MCCS and :XCCS formats are UNSTABLE in that the codes of the same character sequence can be represented in different ways in the same file, depending on whether they are in a 1 byte or 2 byte run.The code for "A" can be a single byte 65 following a preceding 255-0 sequence or by the two bytes 0-65 following 255-255-0. There would be an advantage in changing the :MCCS byte representation to UTF-8 since searching for a string in a stable format can be accomplished by pre-computing the targets byte representing and then doing a byte search. For an unstable format each file character has to be decoded before the match. FILEPOS is thus less efficient for unstable formats and FFILEPOS simply

falls through to the slow case.)

Until now we have not been careful to distinguish the name of the external format (:XCCS) from the name of its character encoding XCCS($). But both :XCCS and :MCCS share the multi-byte representation of character codes but differ in code interpretation. When important to avoid confusion keywords :XCCS/:MCCS will be reserved for the format and the atoms XCCS$/MCCS will denote the distinctive encodings.

Hereinafter and eventually in all source and other files, the string MCCS will replace the string XCCS and :MCCS will replace :XCCS when referring to the Medley internal standard; XCCS will be used only to refer to codes in the true Xerox Character Code Standard and XCCS$ for codes with the swapped dollar sign.

The new MCCS file

The source file MCCS replaces the file XCCS in the loadup. It includes the implementation of the run-coded representation ofmultibyte characters commone to both :MCCS and :XCCS, and it defines the new :MCCS external format and also carries along the :XCCS format. The *DEFAULT-EXTERNALFORMAT* is now initialized to :MCCS (in the file EXTERNALFORMAT).

MCCS puts the following entries on CHARACTERNAMES, to help with setting up the translation functions.

	Circumflex	0,255		[Character names]
▌▌	Currency	0,244		
▌▌	Leftarrow	0,137		
▌▌	Uparrow	0,136		
▌▌	Dollar	0,44		
▌▌	Lowline	0,254	(Lowline is the Unicode name for the XCCS Lowbar at 0,137)	
▌▌			(Note that XCCS has the same glyph as Underline at 0,314)	

Thus, (CHARCODE Leftarrow) produces the code that is interpreted as Left arrow in MCCS, even though that code is interpreted as Lowline in XCCS$ and modern Ascii/Unicode. That code appears in Medley source code (almost always in record create and Clisp binding expressions) and there the left arrow is the intended interpretation.

Note that reading an actual XCCS$-coded plaintext file with the :MCCS external format may not give the proper Medley internal codes—▌▌intended low lines would appear as left arrows. for example.

It turns out that none of our legacy display fonts assign characters according to the MCCS standard. Gacha comes closest, but the Alto text fonts Timesroman, and Helvetica, for example, agree with MCCS but only with respect to the non-control 7-bit Ascii part of the code space. Timesroman and Helvetica assign glyphs to some of the control codes and to some of the higher codes in character set zero. Our so-called NS fonts (Terminal, Modern, Classic, etc.) have dollar in its MCCS position (and so diverge from true XCCS) but the circumflex, lowline, left arrow, and uparrow are not in their XCCS positions. And of course the special fonts HIPPO, CYRILLIC, MATH, and SYMBOL are completely unrelated to either MCCS or XCCS.

The file MCCS includes translation functions for mapping into MCCS the characters that are interpreted with respect to the encodings of particular font familes. This is based on mapping tables that have been moved to MCCS from INTERPRESS.

	ALTOTEXT	used by the Alto text fonts (Timesroman, Helvetica...)	
▌▌		(ATOMCODE *ACODE*)	[Function]
▌▌	XCCS$	used by the NS fonts (Terminal, Modern...).Like XCCS but $ is at 0,44	

	(X$TOMCODE *XCODE*)	[Function]
GACHA	only ^X maps to Lowline	
	(GACHATOMCODE *GODE*)	[Function]
SYMBOL	used only by the symbol font	
	(SYMBOLTOMCODE *SCODE*)	[Function]
MATH	used only by the MATH font	
	(MATHTOMCODE *MCODE*)	[Function]
SIGMA	used only by lispusers/EQUATIONS ?	
	(SIGMATOMCODE *SCODE*)	[Function]
CYRILLIC	used only by the Cyrillic font	
	(CYRILLICTOMCODE *CCODE*)	[Function]
HIPPO	used only by the HIPPO (Greek) font	
	(HIPPOTOMCODE *HCODE*)	[Function]

All of these functions return NIL if the input and output codes are the same. Our legacy font directories contain a few other fonts with idiosyncratic names (ARROWS, DANCER, CARDSTWO...); more work needs to be done to determine whether/how they map to MCCS.

With the cleanup of FONT and the introduction of MEDLEYFONTFORMAT font files, each font has a FONTCHARENCODING property that indicates the character encoding for that font (one of the tags in the list above, plus MCCS). There is also a FONTTOMCCSFN property filled by the function, again taken from the list above, that maps characters in the font's encoding to their corresponding MCCS codes. If the font's FONTCHARENCODING is not MCCS, the function for its particular encoding is returned by MCCSMAPFN:

(MCCSMAPFN *FROMENCODING)* [Function]

This returns, for example, the function HIPPOTOMCODE for the HIPPO *FROMENCODING* but NIL for MCCS. When the font of an imagestream (display or hardcopy device) is changed, the function stored in the new font can be applied to individual characters to produce their MCCS codes. A hardcopy stream can then apply its own function to translate the MCCS codes to its output character encoding. Interpress would apply the function MTOXCODE, HTML would apply MTOUCODE (Unicode).

The function MCCSMAPPAIRS applies those same translation functions to produce a list of translation pairs describing how to rearrange the location of characters in a font. The function MOVEFONTCHARS (in FONT) can use that mapping to move the information for a given character to the corresponding MCCS position, and the FONTCHARENCODING can then be changed to MCCS.

(MCCSMAPPAIRS *FROMENCODING NOIDENTITY)* [Function]

Returns a table that maps all character set 0 codes in *FROMENCODING* to their MCCS counterparts, including converting to slugs (for the display, black boxes) any *FROMENCODING* codes that are undefined in MCCS. The mappings from a code into itself are excluded if NOIDENTITY.

MCCS defines the following functions to convert between MCCS codes and true XCCS, for use in reading and writing true XCCS (but not exactly :XCCS) files.

(MTOXCODE *MCODE*) [Function]
(XTOMCODE *XCODE*) [Function]

Finally, note that the codes for Japanese and Chinese characters are the same for XCCS and MCCS, so that no additional translations are required for the Japanese external formats defined in library>JAPANESE. But there are situations where Kanji/Chines characters should be given special treatment: it is useless to apply the boldify/italicize algorithms to Kanji bitmaps, and there is no reason for other font-coercion algorithms to copy entire Kanji character sets from one font to another. The Japanese and Chinese character sets are recognized by the predicates

(KANJICHARSETP *CHARSET*) [Function]
▌▌(CHINESECHARSETP *CHARSET*) [Function]

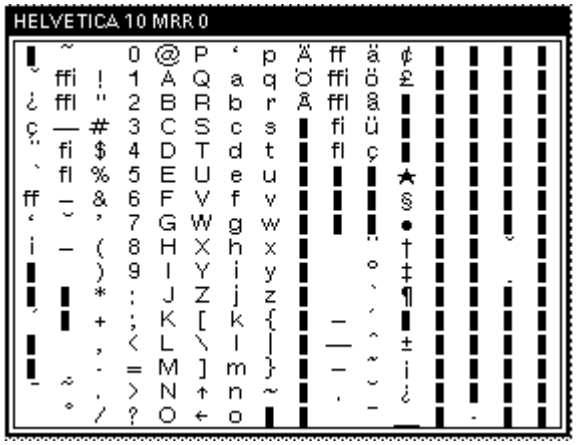
These return *CHARSET* if it is an MCCS character set with Kanji or Chinese characters, otherwise NIL.

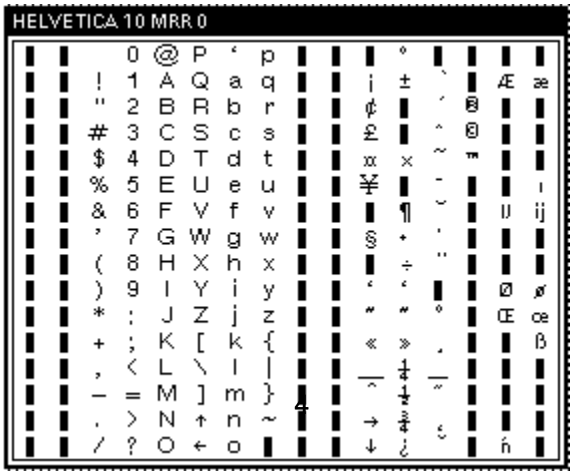
MCCS display (and potentially other) fonts

There are now tools in FONT that allow the information about individual characters (bitmaps, widths, etc) to be moved from one code position to another in the same font or from one font to another. Those tools have been applied to create Medley-format files containing MCCS versions of the Alto and NS text fonts. The new fonts are "character-complete": if there is information about any particular MCCS character in any of the fonts, then every character in each of those fonts has information about that character. The CHARCOERCIONS table for the font-device specifies the preference order for finding fill-in information for missing (slug) characters.

Thus, Gacha starts with characters only in the non-control 7-bit Ascii region of character set 0, every other charset-0 character is an empty slug and every other character set is similarly empty. But following the coercion table, replacements for Gacha slugs are first gathered from Terminal, and if not there, from Modern, and ultimately from Classic. Timesroman is filled in directly from Classic, its closest style match, etc. At the same time, if a font has information about a character but in the wrong code position, the glyph/width is first moved to its proper MCCS position in before other fonts are examined.

The MCCS fonts that result from this process will have all and only the same black boxes (because there are many MCCS character codes which none of our current fonts instantiate). And they will show equivalent glyphs for every character that is known to any of the fonts. Switching the font that a character is displayed in will show an equivalent glyph (or black box), even if stylistically different. The difference for character-set 0 can be seen by comparing the legacy Helvetica 10 font in the first EDITFONT image with the MCCS font in the second.





The 8-bit region of the MCCS font is still very ragged, but this is the best we can do with our current inventory of fonts. In a later phase we can fill in the gaps from BDF-derived fonts, or even wholesale replace all of our fonts with BDF fonts that have been recoded to MCCS.

Even the idiosyncratic fonts (Symbol, Math, Hippo) have been made more compatible with the MCCS standard. Their idiosyncratic codings all reside in character-set 0, but the translation functions map those codes to MCCS position in higher character sets. In the MCCS versions, those characters are left in character set 0 but also copied to their MCCS positions, after which the remaining slugs in all character sets are filled in from other fonts. Thus for the Hippo (Greek) font, the Beta character will appear if printing either "0,101" or "46,101" and printing 357,127 will produce the union symbol.

Deployment of MCCS fonts

Given the availability of MCCS-modified fonts, the question arises as to whether and how they should be included in the system. The goal is to have more characters and strings show up with the intended glyphs while minimizing the risk of damaging the appearance of some characters in some contexts.

Medley source files: The appearance of characters in Medley source files depends on the font profile. Currently the intended glyphs are displayed in the context of (FONTSET 'STANDARD), which installs Gacha and the Alto text fonts. But users who want to see characters outside of character set 0 will likely run with (FONTSET 'NS), and they will see underscores and circumflexes instead of the intended arrows. After the MCCS fonts are deployed and if no character codes are translated, the intended glyphs will appear for both profiles, differing only in stylistic variations.

The translation of character codes for Medley source files is governed by the explicit or defaulted external format of their DEFINE-FILE-INFO expressions. DEFINE-FILE-INFO has been choosing :XCCS as the default format, but it is safe to change its definition in BOOTSTRAP so that it chooses :MCCS instead. The result is that there will be no code translations when a defaulted Medley source file is read. On the other side, it is safe to change PRETTYDEF in PRETTY so that it recognizes :MCCS in a file's reader environment as the new default, and does not include it in the DEFINE-FILE-INFO expression. Code translations will continue, of course, for files with :UTF-8 or other external formats.

Plain text files: An external format can be specified when a plain text file is open for reading or display. Depending on the file's format, character codes will or will not be translated to MCCS. The implicit default format has been :XCCS, but it is proposed to change that to :MCCS. If a file contains any of the codes that differ between the two encodings, the codes will be translated if :XCCS is specified explicitly in the OPEN call. If the previous intended appearance was font-dependent as well as format-dependent, then further adjustment may be needed.

Tedit formatted files: Unlike plain text files, each of the characters in a Tedit formatted file is associated with a particular font, and the intended glyph for each character is the one that is assigned by its font. Tedit implements the translation of codes in non-NS fonts using the tables that have now migrated to MCCS. At the same time, in order to preserve the appearance, it changes the original font to the stylistically most similar NS font, for example, Gacha to Terminal and Hippo to Classic. After these transformations, the glyphs that appear from typing or that are located by searching will operate in a font-independent way.

When the modified-MCCS fonts are installed, after the codes are translated at file-opening, there will be no need for the second step of font renaming (except perhaps for the idiosyncratic fonts). The codes and fonts will both be consistent with respect to MCCS. However, if the file is saved after the code ttranslation, it will be marked to say that the translation has already taken place and so will not be repeated when the new version is open. If the new version is opened in old environments, without the MCCS fonts or without the updated Tedit code, the mark will be ignored but the intended appearance is not guaranteed.

Programmed printing to image streams: Strings and atoms do not make explict the encoding of their

characters. Existing programs that print to an output display stream expect the characters to be rendered in the font that is installed (DSPFONT) at the time of the printing. But after the fonts have been updated to MCCS, strings with non-XCCS interpreted codes will not appear as intended. Those strings will have to be located and modified by hand. A search of all known files for the names of the idiosyncratic fonts will help in identifying those strings. Strings printed will be harder to identify, but in the normal Ascii region only the arrows/underscore/caret should be affected for XCCS$ fonts.

The OUTCHARFN's of particular hardcopy devices can assume that it will receive only MCCS codes, and it only needs to apply the conversion function from MCCS to its own format. Interpress for example would apply MTOXCODE and not deal with any other coercions.

MCCS and Unicode

XCCS is important as the intermediary for establishing the translation between internal MCCS and the codes defined in the Unicode standard. This allows us to read and write Unicode-coded text and hardcopy files and also to make use of externally created fonts and other resources. The correspondence tables in unicode/xerox/ define the relationship between Unicode and XCCS codes. The functions in library/UNICODE compile the information in those tables into the mapping files

▌▌ MCCS-TO-UNICODE.TXT [File]
▌▌ UNICODE-TO-MCCS.TXT [File]

▌▌This is accomplished by first constructing the collection of Unicode/XCCS correspondenc es and then applying either MTOXCODE or XTOMCODE to produce the MCCS translation pairs. UNICODE also provides the functions

(MTOUCODE *MCODE*) [Function]
▌▌(UTOMCODE *UCODE*) [Function]

(MTOUCODE? *MCODE*) [Function]
▌▌(UTOMCODE? *UCODE*) [Function]

to translate codes from one scheme to the other. :UTF-8, :UTF-16BE, and UTF-16LE make use of these translations. The ? versions return NIL if there is no mapping for the argument code. (See the updated UNICODE.TEDIT for more information.)

(MTOUTF8STRING MSTRING) [Function]
▌▌[UTF8TOMSTRING UTF8STRING] [Function]

MTOUTF8STRING produces a string containing the sequence of UTF-8 bytes that represent the Unicode characters corresponding to the MCCS characters in MSTRING. UTF8TOMSTRING converts the string of UTF- bytes into a string of the corresponding MCCS codes.

Typing conventions for MCCS codes

Since the early Alto days and despite the labels on modern keyboards, it has been the case that the MCCS code for leftarrow is produced by typing shift-hyphen and MCCS uparrow is produced by typing shift-6. (Shift-4 has always produced the dollar sign.) But then, how are the low line, cirumflex, and currency symbol to be typed, if that is the user's intent? The keyactions in LLKEY are set up so that those codes can be entered by using the keyboard's function keys:

Circumflex	Function 6	(instead of shift 6)	[Keyboard conventions]
▌▌Currency	Function 4		
▌▌Lowline	Function 10	(instead of shift-hyphen)	

MCCS and the underlying operating system

The underlying operating system does not interpret MCCS characters properly, so translation is

necessary so that Medley strings will map to operating system strings with the same appearance and same interpretation. The Shell file devices is initialized with :UTF-8 as its default internal format, so the translation between MCCS strings and UTF-8 byte representations of the corresponding Unicode characters should happen automatically. The MCCS string passed to FORK-UNIX is explicitly transformed by a call to MTOUTF8STRING. Clip board, GITFNS commands, and various Shell operations should thus work as expected.

The translation of strings across the file-name interface has not yet been implemented. Currently file names with non-Ascii characters in Medley will presumably show up with garbage bytes when viewed with Unix tools, and Unix non-Ascii names will have garbage bytes in Medley. This is as it has always been—the only glitch that is likely to be encountered is the fact that underscores are no longer transparent.

This will be addressed in one of the next updates, by wrapping the strings that pass back and forth across the filename interface functions in either MTOUTF8STRING and UTF8TOMSTRING. The appearances of filenames with combining characters will still be different, because of the different sequence conventions. But at least it will be clear what is combining with what.

There may be other operating system interfaces that have not yet been identified and addressed.