記 号 処 理 44－2
（１９８８．１．２２）

Common Lisp における文字オブジェクトの扱いについて

井田 昌之 （青山学院大学 理工学部）

Common lisp での文字オブジェクトに関する仕様について論じている。まず、現在の仕様とその使用上の不具合、関連する電子メール討論とその結果についてのまとめを行い、特に、次の二つの点について明らかにしている。

一つは、Character 型の仕様に関するものである。3属性のうち、フォント属性についてはその存在自身が疑問視されていること、bits属性については不要論もあるが、既に鍵盤仕様とは独立した概念を形成し、使用されていることから支持があることがわかった。

もう一つは、文字に関連する型の間での変換に関するものである。文字型、数値型、文字列型、シンボル型はdisjointと規定されているので、それらの間の変換機能が重要となるが、それらは変則的で、あまり使いやすくないことを指摘した。結論として、coerceの拡張による統合を示し、それをＣＬＯＳで与えている。

A Note on the Character Objects in Common Lisp

Masayuki Ida
Aoyama Gakuin University
1 Morinosato Aoyama, Atsugi, Kanagawa Japan 243-01

This note is on the character object specification of Common Lisp.First, the current definitions and their awkward are described. Second, the related E-mail discussions are summerized. As the results, this note gives the following two points:

One is on the specification of character data type itself. Font-attribute is not supported by the community. Though there are objections for its existence, bits-attribute has its role,independent from keyboard design.

The other is on the conversion among character, number, string and symbol data types, which are defined as disjoint. The current provisions are irregular and asymmetric. The extension of Coerce function is proposed in CLOS as an integrated specification for the conversion.

1. Character Objects in Common Lisp

To handle character objects in Common Lisp, we have several related data types; `character`, `string`, `integer`, `symbol`, and so on. Though Common Lisp explicitly has the character type as a fundamental data type for character objects, CLtL[Guy84] seems to be ambiguous about the details of character manipulations, and the E-mail archives seemed to contain decisions.

This note discusses on the clarification of several issues on characters, and try to find the essential points. This note starts from several common observations, then presents what was done with E-mails, introducing an analysis and the final remarks.

2. Empirical Observations

2.1 `EXPLODE/IMPLODE` questions
Sometimes, a novice to Common Lisp but who has experiences with other Lisp dialects asks why Common Lisp has no `EXPLODE`, `IMPLODE`. This problem arises too, when try to port codes written in other dialects. (`EXPLODE` explodes a symbol-name to a list of each component 'character' of the symbol-name. It is on menues of Maclisp family, Standard Lisp, Franz Lisp, and others. `IMPLODE` is a reverse function. In standard Lisp, `IMPLODE` is called `COMPRESS`.)

Complaints about the lack of `EXPLODE` and friends had been recognized from very early days of Common Lisp. Wholey@cmu-cs-c.arpa sent a song to common-lisp bboard dated thu, 20 Sept. 1984 13:55EDT;

> I ain't gonna hack on Common Lisp no more,
> I ain't gonna hack on Common Lisp no more.
> With its tons of sequence functions, And its lexical scoping,
> I've now begun to like it, But the users are moping:
> "Without `EXPLODE` my life is full of pain."

This song has a relation to a kind of nostalgia to early days, and indicates there might be lots of codes which use `explode/implode`.

To make an advice, we suggest definitions of `EXPLODE`, `IMPLODE` in Common Lisp, or suggest a different but simpler algorithm to avoid the use of `EXPLODE/IMPLODE`. A simple version of `EXPLODE/IMPLODE` is;

```
(defun EXPLODE (symbol-name) (coerce (string symbol-name) 'list))
(defun IMPLODE   (list)    (intern (coerce list 'string))).
(EXPLODE 'abc) --> (#\A #\B #\C)
(IMPLODE '(#\A #\B #\C))  --> ABC
```

To improve these codes to cope with more types as argument, one can rewrite it as,
`(defun IMPLODE-1 (list) (intern (map 'string #'character list))),`
or with format,
`(defun IMPLODE-2 (list) (intern (format nil "~{~A~}" list))).`

But, clearly the above two definitions have different functionality for numeric arguments.

```
(IMPLODE-1 '(70 a 71)) --> FAG  (on Symbolics)
(IMPLODE-2 '(70 a 71)) --> |70A71|
(IMPLODE '(a b c))   --> is an error
(IMPLODE-1 '(a b c)) --> ABC
(IMPLODE-2 '(a b c)) --> ABC
```

(Jeff dalton (AIAI Edinburgh Univ.) sent his Common Lisp codes for franz explode/implode family to comp.lang.lisp at 6 Aug 1987 20:34:54 GMT. Several Common Lisp implementations give a source code of Mac Lisp explode to users as samples.)

This observation around `EXPLODE/IMPLODE` intrudoces us two issues; one is the spec of the character type, the other is the conversion facilities.

2.2. Does the character data type of Common Lisp bother users ?

As summarized in **appendix**, a character data has three components and is disjoint to a number and a symbol. We have several observation around these as follows.
1) how to use bits and font attributes and why they are.
1.1) bits- is truely independent from code- ?
1.2) bits- represents physical keyboard arrangement ?
1.3) font- is alive ?
1.4) Symbolics has style- attribute avoiding the use of font-.
2) how to convert/distinguish among them.
2.1) one can write a code to check the equality of `x` and `#\A` as, `(and (characterp x) (char= x #\A)) ...`, even if they can be substituted by `eq` or other simple predicates.
2.2) character objects cannot be subjects for arithmetic operations. We must rewrite `(+ #/a 1)` (in Zetalisp) to `(code-char (+ (char-code #\a) 1))`
2.3) A comparison of characters with =,>,< is requested to be rewritten to an expression using `char=, char>, ...` And the arguments to char=, char>, and friends must be characters, else error.

2.3. The Kermit7 Case --To be a String of Characters or a Vector of fixnums ?

The author has a project to write kermit7, which is a kermit for Symbolics Genera 7. The first version was the update of LMKERMIT 1.0 which was included in the kermit distribution from U of Columbia. This version is working well on 3620. (LMKERMIT was originally written by Mark David (LMI), George Carrette (LMI), and was ported and modified by Mark Ahlstrom (honeywell) to 3600.)

Since there were lots of incompatible changes between Zetalisp Genera 6 and Common Lisp Genera 7, LMKERMIT (for Genera 6) cannot work on the current Lisp machines and redesign is needed. The major differences which have relation to the portation, are on the syntax change of Flavors and on the character manipulations. As for the character manipulation, lots of code fragments depend on the mixed use of character operations and numeric operations on the same object. For example, we find an inoperative code fragment like;

```
(cond ...
      ((>= c #\sp) ...)
      ((= c quote) ...)  ; check if quote character
      ((= c #o177) ...) ...).
```

There are two selections: 'string of characters' policy and 'vector of fixnums' policy. To worse the matter, LMKERMIT has a lot of `defconstant`s which are referred across several source modules. And some references are as integers, and others are as characters. 'string of characters' strategy need to insert `char-code` and `code-char` frequently to handle code-values as integers. 'vector of integers' strategy will hide every appearance of `#\`-things and insert conversion codes to the textual manipulation phase. (Since I wanted to test the functionality of character manipulation ability in Common Lisp, I decided to use 'string of characters' policy as a base.) There is no panacea.

2.4 Coercion is not generic

The function `coerce` is defined in *CLtL* as a basic provision for the conversion. `COERCE` is defined mainly for sequence and number. But the defintion is irregular and has several exceptions. (See **appendix**.) The next section describes the problems around coercions.

3. Type Conversion -- function `coerce` and several specific functions --

3.1 The Current Provision and the Awkward in the Current Definitions

Common Lisp has a `coerce` function for a kind of type conversion. **Table 1** summarizes the provisions for conversions among `character`,`integer`, `string`, and `symbol`. It is clear that the convention seems to be irregular or asymmetric, especially on the utility of `coerce`. Among the functions in Table 1, we can pick up the role of `STRING` function. `STRING` has odd definition as shown in **appendix**. It was also the starting point of discussions described in **3.2**.

The problems or the awkwards are mainly on the design of the symmetry of the function names. We would start the discussion with the following two observations.
i) `(string x)` is OK. But, `(coerce x 'string)` is illegal.
While, `(character x)` is OK. And `(coerce x 'character)` is OK too..
ii) To convert from a symbol to a string, use `SYMBOL-NAME` or `STRING`. While, to convert from a string to a symbol, use `MAKE-SYMBOL` to an uninterned symbol, or use `INTERN` to an interned symbol.

Table 1. Possible conversions among character, string, symbol and integer

type of conversion	provided functions	coercion
character -> string	string	X
character <- string	coerce (if the string has only one char.)	O
character -> symbol	(intern (string *ch*))	X
character <- symbol	coerce (if pname length is 1)	O
character -> integer	char-code, char-int	X
character <- integer	code-char (zero font-, zero bits- attrib.) int-char (any font- and any bits-)	O
string -> symbol	intern, make-symbol	X
string <- symbol	string, symbol-name	X
string -> integer	(char-code (coerce *string* 'character))	X
string <- integer	(string (code-char *integer*))	X
symbol -> integer	(char-code (coerce *symbol* 'character))	X
symbol <- integer	(intern (string (code-char *integer*)))	X

3.2 Discussions on Coercion in Common-Lisp E-mails 1986

The awkward listed in **3.1** were discussed already in Common-lisp E-mails. The author checked the 10M bytes E-mails on disk. The discussions around `coerce` were almost in 1986, and not much in 1985 or before. The sequence of our concern started by a mail of fateman@dali.berkeley.edu, dated Fri, 16 May 1986 15:40 PDT as follows.

1) fateman@dali.berkeley.edu fri 16 may 1986 15:40 PDT:
This mail describes the same thing as **3.1 i)**.

2) jeff@aiva.edinburgh.ac.uk sun 18 may 17:17 GMT
... 'string' applied to a sequence of characters gives an error, typically saying the sequence can't be coerced to a string, but 'coerce' will in fact coerce it...

3) gls@think.com, Mon 19 may 1986 12:20 EDT
Research shows that the Common Lisp archives up to a couple of months ago contains 18 messages that mention COERCE. None explicitly addresses this issue, but the general tone of the messages is one of conservatism. I now remember that this issue was tied up with the design of the sequence functions. There was real resistance to letting symbols be treated as general sequences, and so the general decision was made that string-specific functions would accept symbols, but general sequence functions would not. ... To check his talk, **3.3** shows all the early discussions on coerce the author can find.

4) fahlman@c.cs.cmu.edu Mon, 19 May 1986 20:44 EDT
... I would not object to generalizing coerce to handle some of the additional cases that people imagine it ought to handle.

5) cfry@oz.ai.mit.edu, Tue, 20 May 1986 03:21 EDT
... Coercion is a powerful, easy to remember concept. I think it should be extended as much as possible. ... :

```
(coerce #\a 'string) => "a"
(coerce 123 'string) => "123"
(coerce #\a 'integer) => (char-code #\a) ; explicitly not provided CLtL p52.
```

It appears that the only reason is that no one could decide on using char-code or char-int for the conversion so they chose not to do it at all. This reasoning is odd. Pick the most frequently used way, document it, and permit it. Same argument for coercion of numeric types. Further out extensions might be:

```
(coerce #'foo 'compiled-function) => returns a compiled function object
...
(coerce string-1 'pathname)
(coerce bit-vector-1 'integer) ...
```

Undoubtedly there are other coercions which would make sense. ... Users would save a lot of manual searching if coerce was extended.

6) Eliot@umass-cs.csnet, Tue 20 May 1986 15:31 EST
Coercing symbols to stings is fine, as long as NIL is treated as the empty SEQUENCE, rather than as a symbol.

7) REM@IMSSS, 19 May 09:34 PST referring to the mail of gls saying that "COERCE was limited to ... sequence and numerical coercions". This is one of the bad points of many LISPs, including CL, functions that are miss-named or otherwise don't do what you'd expect from their name. ... I hope the international standards committee will fix this kind of problem so programmers reading somebody else's code can have the meaning apparent in most cases form general programming tradition rather than having to constantly check the manual to see if the function does what it seems to say it would do.

8) DCP@scrc-quabbin.arpa, Wed, 21 May 1986 16:45 EDT,
Does (coerce *symbol* 'string) return
(symbol-name *symbol*), or (format nil "~S" *symbol*),
or (format nil "~A::~A"

(package-name (symbol-package *symbol*)) (symbol-name *symbol*))
or what ? If this weren't enough, added in with my personal views of style and functionality, for me to want to help veto this coercion, the special casing of NIL certainly would. Programs should reflect their meaning. A string is a sequence, a symbol is not. Why shouldn't
(coerce :ascii-EOT 'integer)
work? The answer is that the requested behavior is not a coercion between compatible types, it is a functional translation between human understandable names for the ascii control characters and the integers that are their corresponding values.

We found that there is a possibility to extend the semantics of coerce to cope with more generic types. It should be noted that the two key designers of Common Lisp mentioned their opinions, and they do not always be against to the extension. The discussion was stopped at this point and we cannot find their continuation yet. We find from this story that
1) If we don't care about the package, we may extend the coerce to the function from a symbol to a string,
2) If we are free to care about the font- and bits- attribute, we may extend the coerce to include the function from a character to other types.

3.3 Early discussions on coercion

The following sequence was picked up from the archives. They are almost all the meaningful talk the author can find. They were in 1983, one year before *CLtL* was published.

1) Guy Steele, dated 27 May 83 01:23:14 EDT, (in the summary of the discussion with DLW and moon upon the laser edition update)
a) No. 38: it is noted that DLW said "coercing of characters to integers should probably be allowed too." and this point was marked as {controversial}.
b) Moon's comment. "if (string x) <=> (coerce x 'string) exactly, say so. Both of these should coerce a character into a 1-element string; neither says so now. The old argument against this, that people might expect string of a number to give not numbers." and Guy Steele agreed.
N.197: {gloss} string is required to signal an error if its argument is not a string and not a symbol. This is wrong; it should convert a character to a one-character string. Also it says that string will not convert a sequence of characters into a string, but coerce will. This might be all right, except under coerce it says that when coercing to a string the implementation is allowed to return a more general type. ... Also the coerce writeup doesn't say anything for or against coercing from a character to a 1-long string. {controversial} {will clarify}

2) Fahlman, dated Sat, 28 May 1983 22:34 EDT;
At least, I would like you to consider dropping these. ...
Page 38: Coercing of characters to integers is not a very useful operation in portable code if the language spec does not specify the correspondence (and Common Lisp does not). Do you still want to propose that we add this coercion? I'm not particularly opposed, I just worry that users will find it too easy to inadvertently do non-portable things if given this coercion.

3) Moon, date: sat, 28 May 1983, 22:40-EDT
I don't think coercion of characters to integers should be allowed, because how do you know whether you want the char-code or what. Dan was probably just stuck in our old mindset from not having character objects on the Lisp machine. Coercion of characters to strings, however, I believe is useful.

4) Daniel L. Weinreb, Date: Tuesday, 31 May 1983, 15:40-EDT
... It's OK with me if COERCE doesn't coerce characters to integers. However, I strongly suggest putting in a clarification note to that effect... Or maybe a Rationale note saying that it doesn't because it wouldn't be clear what to do about the char-code versus whole character, and telling you to use the particularly function instead. This note is for the benefit of users more than of implementors.

5) Scott E. Fahlman, Date: Wed, 1 Jun 1983 01:47 EDT < referring 4)>
It's OK with me if COERCE doesn't coerce characters to integers.
However, I strongly suggest putting in a clarification note to that ...
<< I assume Guy will do this. >>

As far as we can see from this talk, that the process of making a coercion between characters and integers be restricted such that char-to-integer conversion is not provided, while integer-to-char is. The coercions from characters to integers are purposely not provided; char-code or char-int may be used explicitly to perform such conversions (See **Appendix** for the definitions of char-code and char-int). The difference between char-int and char-code is on the treatment of *font* and *bits* attributes. If these two attributes have no significant meaning and are ignored by everyone, we can make the story much simpler. (And **4.** describes at least font- is not alive).

4. Problems on the Components of Character Object --- Font- and Bits- attributes ---

The most hot discussions were E-mails on the three months from May to July 1986, just after the discussion on coerce, which was summarized in **3.3**.

The discussions were triggered by, (but not guided by), the mail of M.Ida about the kanji and long-char issue dated Fri,30 May 86 14:55 JST and continued under the subject title "Re: long-char, kanji". The series ended with a kind of conclusion that bits-attribute is useful, but no one claims font- is needed.

Here is the series of the discussion:

1) Date: Fri, 30 May 86 14:55:14+0900, From: a37078 (Masayuki Ida)
Subject: long-char, kanji
. . .
The basic idea of my draft:
add long-char, or extended-string-char.
which is needed to represent multi-byte characters.
...
The basic issues:
Is the long-char a subtype of character ?
Is the long-char a subtype of string-char ?
What is the relation between standard-char and long-char ?
Can a vector of long-char be a component of a string ?
If the long-char is separated from string-char, it should have font-attribute or no?

--- Selection 1 ---

```
  make long-char be a subtype of string-char, i.e. string-char > long-char.
  long-char and standard-char are disjoint.
--- Selection 2 ---
  make long-char be a subtype of character type, i.e. character > long-char.
  and, string-char and long-char are disjoint.
--- Selection 3 ---
  make standard-char be a subtype of long-char,
       i.e. string-char > long-char > standard-char.

Possible consequences due to the above selections

Selection 1:
 long-char (2 byte or more) and standard-char(1 byte) can be mixed in a string.
--> It seems to be very heavy for general purpose machines,
    to support ELT, LENGTH etc. correctly.
    And user may confuse on writing software.
--> add string-normalize function.
 (string-normalize x) means,
   if x is purely composed of standard-char then return x.
   if x is purely composed of long-char then return x.
   if x is mix-composed of subtypes of string-char,
     then if all the characters of x can be representable in standard-char
            then each character is converted to the representation in
              standard-char,
            elseif there is at least one character which can only be
                   representable by long-char,
              then all characters are converted to long-char representation.
   else error.
Selection 2:
 New problem will come. That is,
  Can long-char type have non-zero value for char-font and char-bits ?
--> I feel the answer should be "NO".
--> (vector long-char) can not be "string", because string is
 (vector string-char) and string-char and long-char are assumed to be disjoint.
--> Need another type of string, say long-char-based-string,
  which is parallel to string, but is disjoint to string.
```

Though my stress is on the kanji extension, major interest seemed to be in the discussion of the char-font and char-bits issue described in 'Selection 2'. An immediate reply from Moon was posted describing Symbolics' design. In his design, it became clear that Lisp machine has two string types, one is a Common Lisp conformed one, and the other is an extension in which each component character can be an arbitrary one. The later means each component character can have its own font- and bits- attributes.

2) Date: Fri, 30 May 86 16:58 EDT, From: "David A. Moon"
Subject: long-char, kanji
To: Masayuki Ida <a37078%ccut.u-tokyo.junet%utokyo-relay.csnet>

```
Here's how the character and string data types are organized in the
Symbolics system . . .
  . . .
  FAT-CHAR is a subtype of CHARACTER
  FAT-CHAR and STRING-CHAR are an exhaustive partition of CHARACTER
  THIN-STRING and FAT-STRING are an exhaustive partition of STRING
  THIN-STRING is (VECTOR STRING-CHAR)
  FAT-STRING is (VECTOR (OR FAT-CHAR STRING-CHAR))   . . .
It's not true in our system that any character whose bits and font are
zero is a STRING-CHAR, and it's not true that any STRING is a VECTOR of
STRING-CHAR. . . .
```

His message pointed out that a vector of characters, not only of string-chars, is a string.

Scott Fahlman who chaired the E-mail discussion at that time, recognized the problem as one of the important issues to be discussed, and sent the next mail. This mail pointed out the two issues; one is on the extended character sets, and the other is on the 'Fat-string'. And noticed as "Fat characters and strings should be an optional language feature: an implementation does not have to support these, but if it does, it should do it in the standard way. (We can specify some marker that is put on the *features* list if and only if fat characters are supported.)"

He addressed the extended character sets issue as a problem for international standardization noticing "I assume that any Lisp that does not support fat characters will not do well in the Japanese market, ..." (this issue is not the main subject of this note, so the author does not present far more things here).

And he addressed the 'Fat string' issue as a problem upon the clean up things saying " we should define some notion of fat characters and the strings to hold them, ...". But he also suggested the issue should be optional as "The specification of fat characters must be done in such a way that currently legal implementations that do not support them can be left as is; implementations that do support them must be able to do so without penalizing users of normal non-fat strings, either in speed or storage space.", taking care of general machines which may degrade their performance without thought-out design.

The main problem for 'Fat-character' is, as Scott Fahlman said in his mail, on the relation between the *CLtL* defined string-char which cannot have non-zero font and bits and the Fat-Char which can have non-zero font and bits. The Symbolics spec says that Fat-Char and String-Char form an EXHAUSTIVE partition of Character.

He suggested two possible solutions as follows:

3) Date: Fri, 30 May 1986 22:41 EDT, From: "Scott E. Fahlman"
Subject: long-char, kanji

```
  . . .  Two solutions are possible:

First, we could alter the type hierarchy as Moon suggests, and begin to
encourage implementations to exercise their right to have zero-length
font and bit fields in characters.  A lot of us have come to feel that
these were a major mistake and should begin to disappear.  (We wouldn't
legislate these fields away, just not implement them and encourage code
developers not to use them.)  An implementation that does this can have
Fat-Strings with 16-bits per char, all of it Char-Code.

Alternatively, we could say that Fat-Char is a subtype of Character, with
Char-Bit and Char-Font of zero.  String-Char is a subtype of Fat-Char,
with a Char-Code that fits (or can be mapped) into eight bits.  A
Thin-String holds only characters that are of type String-Char.  A
Fat-String holds Fat-Chars (some of which may also be String-Chars).  If
you want a vector of characters that have non-zero bits and fonts, then
```

you use (Vector Character). ...

In the next mail, he threw a ball to everyone on the list as for the change or an extension of the current definition.

4) Date: Sat, 31 May 1986 21:10 EDT, From: "Scott E. Fahlman"
To: "Robert W. Kerns" <RWK@SCRC-YUKON.ARPA>
Subject: long-char, kanji
. . .
I also agree that the current definition of characters, with their bit and font attributes, is a total mess, but it's one that we can live with. I'd love to make an incompatible change here and clean everything up, but we have to move very carefully on such things. There's a lot of code out there that might be affected. We should probably begin with a survey of who would be screwed if char-bits and char-fonts went away.

He triggered the discussion on the real needs for font- and bits- attributes.

5) Date: Mon, 2 Jun 86 08:37:37 MDT, From: shebs%utah-orion%utah-cs.arpa
Subject: Re: long-char, kanji
Didn't you do an informal survey a while back on who actually used the standardized bits and fonts in characters? I believe the consensus was that nobody used them, ... This mail seems to say that Stan Sheb want to flush these spec.

6) Date: Mon, 2 Jun 1986 11:16 EDT, From: "Scott E. Fahlman"
Subject: long-char, kanji: referring to the above 5)
. . . Suppose we were to change the standard to eliminate the Bit and Font fields in characters. (Such fields, along with other attributes such as "style", would be allowed as extensions, but Char-Bit and Char-Font would no longer be part of the standard language.) Would anyone be screwed by this? Would anyone even be unhappy about it? ...
Scott explicitly noted that the possibility of the elimination of bits and font.

7) Date: Mon 2 Jun 86 09:53:28-PDT, From: Evan Kirshenbaum <evan%SU-CSLI.ARPA>
Subject: Re: long-char, kanji
... My vote is for leaving at least bits (and probably font) in the language. ...
He also described with his experiences as an editor a while back, that bits-attribute is useful since he could insert a mark like "added since the last screen refresh" and so on.

8) Date: 2 Jun 86 20:53 PDT, From: fischer.pa%Xerox.COM
Subject: Re: long-char, kanji . . .
In this mail, Xerox stated that bits- and font- attributes are not used.

9) Date: Tue, 3 Jun 86 11:14 EDT, From: "Scott E. Fahlman"
To: Evan Kirshenbaum <evan@SU-CSLI.ARPA>
Subject: long-char, kanji . . . < referring 7) >
Anyway, the question was not whether you ever want to use the notion of "fonts" and "bits" in some generic sense; the question was whether you currently use the Char-Font and Char-Bits facility as currently defined in the language. If people haven't made a lot of use of these in their current form, we are free to think about whether there is some better way of providing the comparable functionality, and how much of the better way ought to be in the standard part of the language. If people have made a lot of use of this facility, we're probably stuck with the status quo, though all the implementors can conspire to make these fields of zero length. ... In this mail, Scott asked whether there are applications actually using font- and bits- or not. This mail shows the basic principle of cleaning up. That is, if the subject spec is actually used by the users, the attitude becomes conservative. He also mentioned he wondered about the meaning of style- attributes.

10) Date: 3 Jun 1986 1043-PDT, From: ccut! Shasta! Rem@IMSSS
Subject: Flush bits from standard

As somebody said, the semantics of bit attributes on characters is dependent on the implementation, therefore no portable program can be written to effectively use them. In general, when a feature can't effectively be used by portable programs, it shouldn't be considered part of the portable standard. ...
Somebody complained about the need to store arbitrary keystrokes in characters. I don't see how this is an argument for portable bit attributes. The keyboards on each machine are different, and thus different numbers of bit attributes will be needed on each machine, and their semantics will be different too.
. . . I vote for stripping bit&font attributes out of the language, but would accept carefully making them as vague as possible ... and documenting just the hook in the manual, saying the details are implementation dependent, ...

This mail also proposes a two level notation of the manual. One is "exactly specifies the semantics" and the other is "merely specifies the syntax and general idea of semantics".

11) Date: 3 Jun 86 15:22 PDT, From: nuyens.pa%Xerox.COM
Subject: re: long-char, kanji
This is just a note to expand on Fischer's info about the Xerox corporate character code standard, ...

This mail describes the Xerox NS system more clear. The points are
1) Xerox think *string* as vector of *characters*, not of *standard-chars*, though Xerox has NO bits- and font- but a kind of style- attributes.
2) Strings are represented as homogeneous simple vectors of thin (8 bit) or fat (16 bit) characters. The difference is transparent to users.
3) Xerox has a string-normalize function.
4) Rendering characters (only included in output image) are different from graphic character codes.

12) Date: Fri, 6 Jun 86 16:34:09+0900, From: a37078 (Masayuki Ida)
Subject: Re: long-char, kanji: referring 11)
...
> ... In particular, since we allow fat characters in symbol print
>names, we use an equivalent of Ida's string-normalize function to
>guarantee unique representation for hashing.
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
This is the most important decision point, I think.
. . .
I like "style" idea also. I don't want to use font. ...

This mail stated 1) a thin-character 'ᴀ' and a fat-character 'A' are different, 2)

font-attribute idea should be flushed from the spec., 3) More than 16 bits may be needed for char-codes in the future as an international standard.

13) Date: Wed 4 Jun 86 11:20:28-MDT, From: SANDRA <LOOSEMORE%UTAH-20.ARPA>
Subject: a standardization proposal
. . . I would like to see things in Common Lisp divided into two distinct categories: (1) things that *every* implementation *must* provide to call itself "Common Lisp"; and (2) things that an implementation need not provide, but for which a standardized interface is desirable. Moreover, I would like to see things in category (2) given standardized names which can be present in *features*, so that you can readily tell whether or not the implementation supports that feature. This approach would also be useful for defining various levels of CL subsets. ... This proposal gave us an interesting key but no one seemed to follow this idea yet.

14) Date: 19 Jun 86 12:32 PDT, From: Masinter.pa%Xerox.COM
Subject: Re: Why aren't char bits portable?
In-Reply-To: KMP%SCRC-STONY-BROOK:ARPA's message of Monday, June 9, 1986 12:23 pm <the author cannot find the mail from his archives>
Char-bits aren't portable because they are a property of the physical keyboard of the machine, and, most keyboards don't have bits that correspond to them. ...
The best thing that a "portable" program can do to cope with the variety of keyboards is twofold:
a) stick as much as possible within the set of standard characters
b) mark the mapping of keys to characters as a clear implementation-dependent part of programs that have to go beyond the standard characters.

15) Date: Fri, 20 Jun 86 10:37 EDT, From: "Daniel L. Weinreb"
Subject: Re: Why aren't char bits portable? <Replying to 14>
I agree with everything you say in your message, except that I would modify the conclusion somewhat. I believe that the intent of including "bits" in the standard is to allow for the fact that some of the "bits" are gaining some amount of currency as ad hoc standards. In particular, there are now several terminals on the market that have a "Meta" key, which is used by many Emacs-family editors. ...
A program that wants to be universally portable, of course, cannot depend on the presence of such key on the user's keyboard. However, it is not hard to imagine a program that wants to be portable, but also wants to allow any user who has a Meta key to take advantage of it. (Indeed, many Emacs-family editors have this property.)

I believe the intention of the "bits" feature was to help out such programs. While I'm not sure that all the details of the "bits" feature in the standard are ideal, nevertheless I wanted to point out that the entire feature is not necessarily bankrupt.

The author realizes the utility of Meta-key, even if it is called alt- key after familiarizing EMACS. Furthermore even if the keyboard has no meta-key, we can enjoy the function of meta-key, like in emacs. This point was also stressed by K. Pitman in the next mail. And the series of discussion ended by his comment saying bits- attribute is independent from physical keyboards.

16) Date: Tue, 24 Jun 86 00:35 EDT, From: Kent M Pitman
<KMP%SCRC-STONY-BROOK.ARPA>
Subject: Why are char bits portable? < referring 14) >
. . . ITS Teco and ITS Emacs (which was ported to TOPS-20) uses char bits quite portably, even though huge numbers of Emacs users have only ASCII keyboards. ... The point is that Lisp itself manages this, and any program I write sees "normal" chars with bucky bits properly attached. CL makes no statements about keyboards at all. ... All we say is that there are functions which read characters from streams. How those characters end up being placed on the stream is pretty much left up to the particular implementation.

As long as users of a portable program can get to all the relevant commands, they don't tend to complain that their keyboard (which perhaps has no META key) cannot type META-X. They just don't use META-X -- presumably they get the same functionality some other way. But users who have a META key -do- complain when there's functionality available and a free key that the functionality could be attached to. The current CL strategy for bucky bits allows me the ability to deal with exactly this problem -- attaching things to chars when they're available and not when they're not. Why would you want to deny me the opportunity to do that? ... Also, the presence of bucky bits in CL may help encourage some keyboard vendors to add bits. If we say "let's not use these bits because no keyboard vendors provide them", then I fear that at their next design meeting, they'll say "let's not provide these bits because no languages seem to use them". Someone's got to take the first step. We already have. I see no reason to back off on it if people (eg, me) are using it productively.

5. Final Remarks

As a conclusion of the analysis, the problems are classified into the following three categories and the author proposes for each;

1) Coercion: COERCE should be more generalized. The observations show there are no problem if extensions are fully written out in the details. As a conclusion of the author for the analysis of **3.**, here is an extension by the author to the current coerce definition using in CLOS [ANSI87].

```
(defmethod coerce ((x character) (y 'string)) (string x))
(defmethod coerce ((x character) (y 'symbol)) (intern (string x)))
(defmethod coerce ((x character) (y (member 'fixnum 'integer 'number)))
   (char-code x))
(defmethod coerce ((x string) (y 'symbol)) (intern x))
(defmethod coerce ((x symbol) (y 'string)) (string x))
(defmethod coerce ((x string) (y (member 'fixnum 'integer 'number)))
   (char-code (coerce x 'character)))
(defmethod coerce ((x integer) (y 'string)) (string (code-char x)))
(defmethod coerce ((x symbol) (y (member 'fixnum 'integer 'number)))
   (char-code (coerce x 'character)))
(defmethod coerce ((x integer) (y 'symbol))
   (intern (string (code-char x))))
(defmethod coerce ((x integer) (y 'character))
   (code-char x)) ; redefinition. CLtL defines as int-char
```

The keys are a) ignore char-bits and char-font upon the conversion of characters, b) ignore the package name upon the conversion of symbols.

2) The problem of the components of strings
2.1) How to cope with the needs of 'using strings with integer elements'
This issue left for future research. The author cannot find the only one solution.
2.2) How to cope with the concepts of 'elements of a string can be not only string-chars but also characters'. Basically, string is a pointer array of characters logically. We already cope with kanji-text mixture problems in Japan. So, strings should be expanded to handle Fat-characters with arbitrary styles. It is the next step to define the encoding scheme of bits- and style- in files/stream-texts, consulting actual customs and ISO character representation standards, ... "How can we represent a string with non-zero bits- or style- characters in double-quote syntax ? Can they be portable ? How can we transfer it to the different machines ?". We have experiences with kanji- string transfer through ASCII only machines. Will this experience assist the case ? This talk is our future task to be solved.

3) The problem of the three components of a character data.
The E-mail discussion seemed to have a conclusion that no one like/support/use/offer font- attribute actually. While, several persons agreed bits-attribute is important and should remain though another person said bits-attribute be flushed from Common Lisp. The author suggest to flush font- attribute from Common Lisp specification.

4) The problem of the relation between symbols and strings. This issue has a relation to the role of the package name upon conversion. The author want to suggest that the package name should not have any role upon conversion.

5) As for standardization, severals pointed out that the definition may have several levels. The author think the hierarchy of the spec is the next step and left for future works.

Acknowledgments

The author sincerely express his gratitude to the following individuals;
Prof. Ishida and the stuffs of Computer Centre Univ. of Tokyo who support his communication using CS-net/Junet, The original writers of LMKERMIT codes with which the author consulted, Mr. Shiota (Nippon Symbolics) who assisted the first version of Kermit7, Mr. Maeda (Aoyama Gakuin Univ.) who assisted to develop small tools, and all the participants of the Common Lisp E-mail discussions.

References

[ANSI87] ANSI X3J13 87-002: Common Lisp Object System Specification, 1987
[Guy84] Guy L. Steele Jr. et. al.: Common Lisp the Language, Digital Press, 1984

Appendix: The Basic Definitions

STRING; *CLtL*[Guy84] says in page 299 that "type STRING is identical to the type (VECTOR STRING-CHAR), which in turn is the same as (ARRAY STRING-CHAR (*))."

STRING-CHAR; in page 34, "STRING-CHAR is a subtype of CHARACTER." While, page 23 says "any character whose bits and font attributes are zero may be contained in strings. ... ; this subtype is called STRING-CHAR."

STANDARD-CHAR; in page 34, "STANDARD-CHAR is a subtype of STRING-CHAR," and in page 21, "the COMMON LISP standard character set is apparently equivalent to the ninety-five standard ASCII printing characters plus a newline character."

CHARACTER; in page 23, "Every object of type CHARACTER has three attributes: *code*, *bits*, and *font*." The possible values for *bits*-attribute are not explicitly defined, but, control-, meta-, hyper-, super- are suggested in the examples.

Disjointness: in page 33, " the types CONS, SYMBOL, ARRAY, NUMBER, and CHARACTER are pairwise disjoint."

CHAR-CODE : The argument must be a character object. CHAR-CODE returns the code attribute of the character object;

CHAR-INT : The argument must be a character object. CHAR-INT returns a non-negative integer encoding the character object. If the font and bits attributes of *char* are zero, then CHAR-INT returns the same integer CHAR-CODE would. Also, (CHAR= c1 c2) is equivalent to (= (CHAR-INT c1) (CHAR-INT c2)) for characters c1 and c2. (INT-CHAR is provided for the reversing.)

COERCE : in page 50 and 51,
1) a sequence type to any other sequence type. Elements of the new sequence will be EQL to corresponding elements of the old sequence.
2) Some strings, symbols, and integers may be converted to characters. If *object* is a string of length 1, then the sole element of the string is returned. If *object* is a symbol whose print name is of length 1, then the sole element of the print name is returned. If *object* is an integer *n*, then (int-char *n*) is returned.
3) any non-complex number can be converted to a XXX-float.
4) any number can be converted to a complex number.

STRING function : in page 304, "STRING accepts a string, a symbol, and a string character. When a string, it is returned. When a symbol, its print name is returned. When a STRING-CHAR, then a string containing that one character is returned. Otherwise error. To convert a sequence of characters to a string, use COERCE. (Note that (COERCE *x* 'STRING) will not succeed if *x* is a symbol. Conversely, STRING will not convert a list or other sequence to be a string.) To get the string representation of a number or any other Lisp object, use PRINT1-TO-STRING, PRINC-TO-STRING, or FORMAT."