NS Character Set Issues:

Greg Nuyens. Aug 16,1984

This note describes a proposal for incorporating the NS character set standard into Interlisp-D. This proposal will encompass the following areas:

--implementation of a "long" character data type

It specifically omits the following:

--external file representations. **

--the reworks necessary to fonts, etc. to attain DIG, though this proposal is made with these issues in mind.

Definitions

The following terms are defined for use in the remainder of the document.

character: an instance of the "long" character data type.

byte: an 8 bit unsigned integer corresponding to the current Interlisp-D character.

character set: the (conceptual) array of 256 characters into which all the representable distinct characters have been separated. An example is character set 0 which contains most but not all of the characters currently used in Interlisp. For instance an umlaut is not included in character set 0.

font: A style in which characters are rendered. A font includes (conceptually) an image for every representable character (2^{16} - 256). A font is a member of the cross product of Family, Weight, Slope, Face, Nominal Size, Rotation, and Expansion . An example is TimesRoman 12 Bold Italic Regular Compressed with 90 degree rotation. To contrast character set and font, note that only one character set (0 *?*) contains the greek letter alpha, but every font contains a rendering for that character. In practice few (no?) fonts will contain all the images of all the characters.

Overview

Before discussing how these functions will change, an overview of the scheme for the internal functions:

Unknown IMAGEOBJ type GETFN: SKIO.GETFN SKCP0072 ((.05 13.0 NIL) (TEXT (16.0 . 127.0) ("String descriptor") 1.0 (LEFT BASELINE) (
HELVETICA 8 (MEDIUM REGULAR REGULAR)) ((16.0 125.0 82.0 9.0))
NIL)) ((.1 16.0 NIL) (WIRE ((
64.0 . 127.0) (64.0 . 111.0)) 1)) ((.1 84.0 NIL) (BOX (16.0 95.0 168.0 16.0) 1)) ((.1 84.0
 NIL) (BOX (16.0 111.0 168.0 16.0) 1)) ((.1 16.0 NIL) (WIRE ((96.0 . 111.0)
(96.0 . 95.0))
 1)) ((.05 9.0 NIL) (TEXT (96.0 . 115.0) ("Base") 1.0 (CENTER BASELINE)
(HELVETICA 8) ((84.0
 113.0 24.0 9.0)) NIL)) ((.1 48.0 NIL) (OPENCURVE ((168.0 . 119.0) (200.0
. 119.0) (216.0
. 103.0) (200.0 . 87.0) (192.0 . 79.0) (192.0 . 71.0)) (ROUND 1 7) NIL (NIL
(LINE 30.0
8.0)))) ((.1 4 NIL) (OPENCURVE ((168.0 . 119.0) (168.0 . 119.0)) (ROUND 8
7) NIL)) ((.05 9.0
NIL) (TEXT (136.0 . 99.0) ("Length") 1.0 (CENTER BASELINE)
(HELVETICA 8) ((120.0 97.0 33.0
9.0)) NIL)) ((.05 9.0 NIL) (TEXT (56.0 . 99.0) ("Offset") 1.0 (CENTER
BASELINE) (HELVETICA 8)
 ((42.0 97.0 28.0 9.0)) NIL)) ((.1 16.0 NIL) (WIRE ((40.0 . 127.0) (40.0 .
111.0)) 1)) ((

```
.05  9.0 NIL)  (TEXT  (52.0 .  115.0)  ("Type")  1.0  (CENTER  BASELINE)
(HELVETICA 8) ((40.0  113.0
 24.0  9.0)) NIL))  ((.05  9.0 NIL)  (TEXT  (28.0 . 115.0)  ("Ind")  1.0  (CENTER
BASELINE)  (
HELVETICA 8)  ((21.0  113.0  15.0  9.0)) NIL))  ((.1  88.0 NIL)  (BOX  (192.0
55.0 176.0  16.0) 1))
 ((.1  16.0 NIL)  (WIRE  ((220.0 .  71.0)  (220.0 .  55.0)) 1))  ((.1  16.0 NIL)
(WIRE  ((252.0 .
71.0)  (252.0 .  55.0)) 1))  ((.1 16.0 NIL)  (WIRE  ((284.0 . 71.0)  (284.0 .  55.0))
1))  ((.1
16.0 NIL)  (WIRE  ((316.0 .  71.0)  (316.0 .  55.0)) 1))  ((.05  9.0 NIL)  (TEXT
(204.0 .  59.0)  (
"Char")  1.0  (CENTER  BASELINE)  (HELVETICA 8)  ((193.0  57.0  23.0  9.0))
NIL))  ((.05  9.0 NIL)  (
        TEXT  (236.0 .   59.0)  ("Char")  1.0  (CENTER  BASELINE)
        (HELVETICA 8)  ((225.0  57.0  23.0  9.0)) NIL
        ))  ((.05  9.0 NIL)  (TEXT  (268.0 .  59.0)  ("Char")  1.0  (CENTER
        BASELINE)  (HELVETICA 8)  ((257.0
         57.0  23.0  9.0)) NIL))  ((.05  9.0 NIL)  (TEXT  (300.0 .  63.0)  (". .
        .")  1.0  (CENTER BASELINE)
    (HELVETICA 8)  ((292.0  61.0  17.0  9.0)) NIL))  ((.05  9.0 NIL)  (TEXT
        (332.0 .  59.0)  ("Char")
        1.0  (CENTER  BASELINE)  (HELVETICA 8)  ((321.0  57.0  23.0
        9.0)) NIL))  ((.05  9.0 NIL)  (TEXT  (
        200.0 .   151.0)  ("Current  String  Descritptor")  1.0  (CENTER
        BASELINE)  (HELVETICA 10  (MEDIUM
        REGULAR REGULAR))  ((122.0  149.0  156.0  12.0)) NIL)))
        (0 0 410.0 192.0)
        1.0
```

There will be two types of character data, the previously described long and short characters. However, any single string will consist of only one type of character. Thus strings as currently represented, (effectively byte arrays) will continue to exist. However, there will also be arrays of long characters. Any functions which receive arguments of mixed type (e.g. RPLCHAR given an string of short chars and a long char to replace with) will produce results composed of long chars only.

Thus, the current implementation of substrings (as "tails" of strings) will not suffice, since the original string may be coerced upward to a long char string by an operation occurring after a substring has been returned. The plan is to use forwarding pointers in string space and store substrings as <header,offset,length> triplets. Thus when a short char string is coerced upward into a long char string, the original header will have a pointer to the new location. Any substrings returned previous to the coercion will still be valid, since they reference the (now indirect) original header. The offsets will always be scaled by a bytes-per-character value implicit in the type of string (1 for short strings, 2 for long strings).

This scheme guarantees the advertised property of substrings that they are indeed shared tails. That is, a destructive change to a substring will

affect the string. This will be true regard**less of any coercion** of the string that occurs.
Also recognize that any coercions performed on substrings will change the representation of the whole string.

Changes to Interlisp Functions
In previous discussions the two following lists of functions affected by this proposal were identified:
First the functions which deal with the internal representation of character