

faststring.m

COLLABORATORS

| | | | |
|---------------|--------------------------------|-------------------|------------------|
| | <i>TITLE :</i> faststring.m | | |
| <i>ACTION</i> | <i>NAME</i> | <i>DATE</i> | <i>SIGNATURE</i> |
| WRITTEN BY | | February 12, 2023 | |

REVISION HISTORY

| NUMBER | DATE | DESCRIPTION | NAME |
|--------|------|-------------|------|
| | | | |

Contents

| | | |
|----------|-------------------------------------|----------|
| 1 | faststring.m | 1 |
| 1.1 | faststring.m | 1 |
| 1.2 | Introduction | 2 |
| 1.3 | History | 2 |
| 1.4 | About the fstrings | 2 |
| 1.5 | longword-copy | 3 |
| 1.6 | Functions | 4 |
| 1.7 | Functions/Template | 5 |
| 1.8 | Functions/fastString() | 6 |
| 1.9 | Functions/fastStrDipose() | 6 |
| 1.10 | Functions/fastStrMax() | 7 |
| 1.11 | Functions/fastStrLen() | 7 |
| 1.12 | Functions/fastSetStr() | 8 |
| 1.13 | Functions/fastStrCopy() | 9 |
| 1.14 | Functions/fastFstrCopy() | 9 |
| 1.15 | Functions/fastStrAdd() | 10 |
| 1.16 | Functions/fastMidStr() | 11 |
| 1.17 | Functions/fastRightStr() | 11 |
| 1.18 | Functions/fastRightFstr() | 12 |
| 1.19 | Functions/fastStrDub() | 12 |
| 1.20 | Functions/fastFstrDub() | 13 |
| 1.21 | Bugs | 14 |
| 1.22 | Legal stuff | 14 |

Chapter 1

faststring.m

1.1 faststring.m

```
*****
*
*           EModule: faststring.m           *
*           Version 1.2                     *
*
*   faster and/or better string-functions   *
*
*           (c) 1995 Jan Hendrik Schulz     *
*
*****
```

Introduction
Why have I done this?

History
What's new?

FStrings
About the new strings

Functions
The functions

Bugs
+-->

Copyright
Legal stuff and my address

|
+-- ATTENTION: I moved! Read this for my new address!

1.2 Introduction

Introduction
 ~~~~~

All began with a program I wrote, which was very slow in some situations. I used AProf to find out why, and the problem was DisposeLink(). I used it to dispose dynamically allocated strings (with String()) and if the number of strings were a little bit bigger, my program seems to sleep 2-3 seconds.

I began to think about an alternative for DisposeLink(). I decided to use simple byte-arrays, FastNew() and FastDispose() instead of estrings, String() and DisposeLink() wherever possible. But the problem with FastDispose() is, it needs the size of the memory. And the most E string functions only work with estrings and not with simple byte-arrays.

Now I had the idea to create a new string-type with a length and a maximum length entry at negative offset (like estrings). The maxlen-entry makes it possible to use FastDispose() in the string-dispose function without having to know the size of the string when calling this string-dispose function.

And because even these new strings are not compatible with estrings, I wrote some string-handling-functions too. Some of these functions are improved (in speed and/or functionality) against their eststring equivalents, and some of them don't have an eststring equivalent at all.

As I changed my program to use these new functions, the time needed to dispose even high numbers of strings wasn't noticeable anymore. But besides that, the program behaves like before.

## 1.3 History

History

~~~~~

Version 1.2:

- Guide-file changed a bit (I moved again -> new address)
- Version 1.1 was only uploaded to a local BBS, but not to aminet.

Version 1.1:

- I noticed, that I hadn't used the short version of branches where possible ('Bxx.S lab' instead of 'Bxx lab'). Now the module is 56 bytes smaller and the functions are a little bit faster.

Version 1.0:

- first release.

1.4 About the fstrings

About the fstrings

~~~~~

To understand the following descriptions, you should be familiar with strings, estrings and the E string functions. Maybe you should read the E docs (chapter 9B) first.

The string functions of this emodule are all working with a new string type. To distinguish these strings from simple strings (arrays of char) and estrings, I call them fstrings (fast strings). Fstrings and estrings have the following things in common:

- They have lenght and maximum-lenght information at negativ offset.
- They are auto-disposed at the end of the program.
- Like the original E string functions, the fstring functions will handle cases where fstrings tends to get longer than there maximum lenght correctly.
- If somewhere a simple string is needed, you could also use an estring or an fstring (if the fstring is changed: see  
`fastSetStr()`  
`)`).
- If somewhere an estring/fstring is needed you MUST use an estring/fstring.
- If the maximum lenght is n, the fstring/estring is long enough to hold n characters PLUS the 0-byte at the end.

And that are the differences:

- The maximum lenght of an fstring is 65530
- There is of course no 'DEF fstr[10]:FSTRING' (or something like that) to get an fstring, you must use  
`fastString()`  
  - fstrings couldn't be linked (see E docs chapter 9H)
- You can't use an fstring with a function that needs an estring.
- You can't use a string or estring with a function that needs an fstring.
- To allow some of the fstring functions to perform a  
`longword-copy`  
`,` the  
maximum lenght of an fstring is allways something like:  $\text{maxlen}+1 = 4*n$   
(see  
`fastString()`  
`)`

## 1.5 longword-copy

## longword-copy

~~~~~

Normaly, if the contents of a string is copied into another string, it's done byte by byte, to be able to recognize a 0-byte which marks the end of the string. But if the lenght of the string, and so the number of characters to copy, is known BEFORE the copying starts, it isn't necessary to listen for the 0-byte anymore. That makes it possible to use a longword-copy, that copies 4 characters (1 longword = 4 bytes) in one step, to make the copying faster.

But there is a problem with a longword-copy: Every longword contains 4 bytes, and so, the last copied longword maybe contains up to 3 bytes more than needed. To make shure, that even these superfluous bytes fit into the destination fstring,

```
fastString()
```

```
increases the given maxlen-value if
```

necessary to create allways fstrings with a total lenght (maximum lenght plus 1 byte for the 0-byte) of $4*n$ bytes (= n longwords). So, (only) if the destinations string for a copy is an fstring, a longword-copy is possible. But:

There is another problem with a longword-copy: Both strings MUST start at an even address, because (at least) older processors (like 68000) couldn't access longwords on odd addresses. If both strings are fstrings (see

```
fastFstrCopy()
```

), this is no problem, because they allways begin at an even address. And if only the destination-string is an fstring, a longword-copy is only possible if the source-string also starts at an even address.

The following fstring-functions are using a longword-copy:

```
fastFstrCopy()
```

```
(allways)
```

```
fastStrDub()
```

```
(only if sourcestring starts at an even address)
```

```
fastFstrDub()
```

```
(allways)
```

```
fastRightStr()
```

```
(only if sourcestring starts at an even address)
```

```
fastRightFstr()
```

```
(only if sourcestring starts at an even address)
```

1.6 Functions

The functions

```
~~~~~
```

Template

How are the functions explained

```
fastString()
```

```
fastStrDispose()
```

```
fastStrMax()
```

```
fastStrLen()  
fastSetStr()  
fastStrCopy()  
fastFstrCopy()  
fastStrAdd()  
fastMidStr()  
fastRightStr()  
fastRightFstr()  
fastStrDub()  
fastFstrDub()
```

1.7 Functions/Template

All the functions are described like this:

NAME

The name of the function.

SYNOPSIS

How to call the function.

ESTRING

The equivalent estring function.

SPEED

The speed of the function against its estring equivalent. A value of 1 means, both are equal, 2 means, the estring function needs twice the time of the fstring function and 0.5 would mean, the estring function is twice as fast.

Because the speed depends on many different things (like stringlength, memory-fragmentation, processor type, ...) these values are only average values to give you an idea of the speed. In some situations it could be totally different!

FUNCTION

What does the function.

DIFFS

The differences between the fstring-function and the estring equivalent.

SEE ALSO

Where to find further information.

1.8 Functions/fastString()

NAME

fastString

SYNOPSIS

fstr,maxlen:=fastString(maxlen)

ESTRING

estr:=String(maxlen)

SPEED

2 - 3

FUNCTION

fastString() allocates a new fstring with a maximum length of (at least) maxlen. The real maximum length (which is returned as second returnvalue) could be up to 3 bytes longer to make maxlen like: maxlen+1=4*n (see

longword-copy
about this)

The returned fstring is initialised as empty string (fstr='')

DIFFS

maxlen must be: 0 <= maxlen <= 65530

fastString() uses FastNew() to allocate the memory and because of this, a "MEM" exception could be raised.

SEE ALSO

fastStrDispose()

1.9 Functions/fastStrDispose()

NAME

fastStrDispose

SYNOPSIS

fastStrDispose(fstr)

ESTRING

DisposeLink(estr)

SPEED

4 - 5

FUNCTION

The fstring fstr is disposed. If fstr is NIL, nothing happens.

DIFFS

fastStrDispose() uses FastDispose() to free the memory.

SEE ALSO

fastString()

1.10 Functions/fastStrMax()

NAME

fastStrMax

SYNOPSIS

maxlen:=fastStrMax(fstr)

ESTRING

maxlen:=StrMax(estr)

SPEED

not determined, but probably nearly 1

FUNCTION

Returns the maximum length of the fstring. (It's the same value like that returned by

fastString()
as second returnvalue)

DIFFS

none

SEE ALSO

fastStrLen()

,
fastSetStr()

1.11 Functions/fastStrLen()

NAME

fastStrLen

SYNOPSIS

len:=fastStrLen(fstr)

ESTRING

len:=EstrLen(estr)

SPEED

not determined, but probably nearly 1

FUNCTION

Returns the length of the fstring, or to be more precise the length

information which is stored at negativ offset. If you change to contents of an fstring with a function other than the fstring-functions from this emodule, you have to set the new lenght with

```
fastSetStr()
```

.

DIFFS

none

SEE ALSO

```
fastSetStr()
```

,

```
fastStrMax()
```

1.12 Functions/fastSetStr()

NAME

fastSetStr

SYNOPSIS

```
len:=fastSetStr(fstr,len=-1)
```

ESTRING

```
SetStr(estr,len)
```

SPEED

not determined, but probably nearly 1 with len>=0 and <1 with len=-1.

FUNCTION

Sets the lenght of an fstring manually. len must be -1 or

0 <= len <= Min(fastStrMax(fstr), StrLen(fstr)).

If len is out of bounds, it will give unpredictable results.

DIFFS

- If you call fastSetStr() with len=-1 (or simply without giving len) fastSetStr() determines the lenght of the fstring itself by searching the 0-byte at the end of the fstring. The result is the same as with: fastSetStr(fstr,StrLen(fstr)), but it's faster. That's handy if you e.g. passed the fstring to an OS-function which changed the contents (and the lenght) but of course didn't changed the lenght information. Before you could use this fstring again with an fstring function you MUST set the lenght using fastSetStr()
- fastSetStr() returns the lenght (usefull if you call it with len=-1).
- If you give a value <>-1 for len, fastSetStr() not only sets the lenght information at the negativ offset to len, it also performs a fstr[len]:=0. That is usefull if you want to shorten an fstring.

SEE ALSO

```
fastStrLen()
```

,

```
fastStrMax
```

1.13 Functions/fastStrCopy()

NAME

fastStrCopy

SYNOPSIS

fstr, len:=fastStrCopy(fstr, str, max=-1)

ESTRING

estr:=StrCopy(estr, str, max=ALL)

SPEED

0.9 - 1 (see DIFFS and
fastFstrCopy()
)

FUNCTION

The string 'str' is copied to the fstring. The number of copied characters equals the_minimum_of(max, StrLen(str), fastStrMax(fstr)) if max<>-1 and the_minimum_of(StrLen(str), fastStrMax(fstr)) if max=-1

'str' could be a string, an estring or an fstring (but if it is an fstring you should use

fastFstrCopy()
instead).

DIFFS

- StrCopy(estr, str, 0) does nothing (the estr isn't changed at all), but fastStrCopy(fstr, str, 0) sets the fstr to an empty string (fstr=''). I think, that's in most cases more usefull.
- fastStrCopy() returns the new lenght of the fstring as second return-value.

SEE ALSO

fastFstrCopy()

1.14 Functions/fastFstrCopy()

NAME

fastFstrCopy

SYNOPSIS

fstr1, len:=fastFstrCopy(fstr1, fstr2, max=-1)

ESTRING

estr:=StrCopy(estr, str, max=ALL)

(Not really equivalent, because 'str' don't HAVE TO be an estring, but it

gives the equivalent result.)

SPEED

1.5 compared with StrCopy() and

1.6 compared with

```
fastStrCopy()
FUNCTION
```

Like

```
fastStrCopy()
, but the sourcestring MUST also be an fstring.
```

The main difference between fastFstrCopy() and

```
fastStrCopy()
is, that
```

the later one copies the characters byte by byte (to recognize a 0-byte which marks the end of the string) while fastFstrCopy() uses the faster

```
longword-copy
```

. That's possible, because the sourcestring is an fstring and so it starts at an even address and its length (and with that the number of characters to copy) is known BEFORE the copying starts.

DIFFS

- Both strings have to be fstrings.

- A

```
longword-copy
is used.
```

- Plus the same diffs like

```
fastStrCopy()
SEE ALSO
```

```
fastStrCopy()
```

1.15 Functions/fastStrAdd()

NAME

fastStrAdd

SYNOPSIS

```
fstr, len:=fastStrAdd(fstr, str, max=-1)
```

ESTRING

```
estr:=StrAdd(estr, str, max=ALL)
```

SPEED

0.9 - 1 (see DIFFS)

FUNCTION

Same like

```
fastStrCopy()
, but the string is attached to the end of the
```

fstring.

DIFFS

- Returns the new length of the fstring as second returnvalue

SEE ALSO

`fastStrCopy()`

1.16 Functions/fastMidStr()

NAME

`fastMidStr`

SYNOPSIS

`fstr, len:=fastMidStr(fstr, str, pos, max=-1)`

ESTRING

`estr:=MidStr(estr, str, pos, max=ALL)`

SPEED

not determined (sorry)

FUNCTION

The only difference between `fastMidStr(fstr, str, pos, max)` and `fastStrCopy(fstr, str+pos, max)` (note the '+pos') is, that `fastMidStr()` checks if `pos>StrLen(str)` and sets `fstr` to an empty string in this case. If you are shure, that `pos<=StrLen(str)` you should use `fastStrCopy()` like above, that's faster (no checking of pos).

DIFFS

- If `max=0` the fstring is set to an empty string (`fstr=''`) (`MidStr()` with `max=0` gives the same result like `max=ALL`)
- Returns new length of the fstring as second returnvalue.

SEE ALSO

`fastStrCopy()`

1.17 Functions/fastRightStr()

NAME

`fastRightStr`

SYNOPSIS

`fstr, len:=fastRightStr(fstr, str, len)`

ESTRING

none available (`RightStr()` needs two estrings, see `fastRightFstr()`)

SPEED

not available
 (0.4 compared with
 fastRightFstr()
)

FUNCTION

Copies the last 'len' characters of the string 'str' (or the whole string if len>StrLen(str)) into the fstring. If possible a longword-copy is used, but because the length of the string must be determined first to find out where the last 'len' characters start, fastRightStr() is not very fast.

DIFFS

not available

SEE ALSO

"fastRightFstr()"

1.18 Functions/fastRightFstr()

NAME

fastRightFstr

SYNOPSIS

fstr1,len:=fastRightFstr(fstr1,fstr2,len)

ESTRING

estr1:=RightStr(estr1,estr2,len)

SPEED

1.5

FUNCTION

Copies the last 'len' characters of the fstring 'fstr2' (or the whole fstring if len>fastStrLen(fstr2)) into the fstring 'fstr1'. If possible a longword-copy is used. fstr1=fstr2 is allowed.

DIFFS

· Returns the new length of 'fstr1' as second returnvalue.

SEE ALSO

fastRightStr()

1.19 Functions/fastStrDub()

NAME

fastStrDub

SYNOPSIS

newfstr, len, maxlen:=fastStrDub(str, extra=0)

ESTRING

newestr:=StrCopy(String(maxlen:=(len:=StrLen(str))+extra), str, ALL)

SPEED

2.5

FUNCTION

Allocates a new fstring which is long enough to hold the string 'str' and maybe some extra bytes. Then the string is copied into this fstring. This function is very usefull if you get a string from an OS-function and you have to dublicate it because the memory were the string stands is used later for another string (example: fileinfoblock and Examine()/ExNext())

If possible a

longword-copy
is used.

DIFFS

.

fastString()
is called to get the new fstring and so a "MEM" exception
could be raised.

SEE ALSO

fastFstrDub()

1.20 Functions/fastFstrDub()

NAME

fastFstrDub

SYNOPSIS

newfstr, len, maxlen:=fastFstrDub(fstr, extra=0)

ESTRING

newestr:=StrCopy(String(maxlen:=(len:=EstrLen(estr))+extra), estr, ALL)

SPEED

2.5

(1.2 compared with
fastStrDub()
)

FUNCTION

The same like
fastStrDub()

but the sourcestring must be an fstring. That,
makes calculating the maximum length of the new fstring faster.

DIFFS

.

fastString()
is called to get the new fstring and so a "MEM" exception
could be raised.

SEE ALSO

fastStrDub()

1.21 Bugs

Bugs

~~~~

I hope, there are no bugs, but I tried to make the functions as fast and  
as small as possible. They are written 100% in assembler and I used some  
inter-function-jumps to reduce the size.

All the functions are tested with edbg and enforcer, and I already used  
some of the functions in a program without problems, but maybe there is  
nevertheless a bug left, who knows?

BTW: If you call the functions with (length) values out of range, they  
definitive will produce unpredictable results!

## 1.22 Legal stuff

Author

~~~~~

(ATTENTION: I moved!)

snail mail: Jan Hendrik Schulz
Elsässer Str. 19
22049 Hamburg
Germany

e-mail: schulz_j@informatik.fh-hamburg.de
schulzjan@dame.shnet.org

Copyright

~~~~~

All the faststring-functions, the faststring-emodule and this guide-file  
are (c)opyright 1995 Jan Hendrik Schulz

Using and redistributing of the emodule and this guide is allowed as long

---

as the following points are observed:

- The emodule and this guide must be unchanged when redistributing.
- You must allways redistribut both (the emodule and this guide) together (as long as the emodule is not compiled into a program).
- You are allowed to use this emodule in your programs for free and then you can do with your program whatever you want.
- You dont't have to include a note in your program or its docs about the using of this emodule and that it is copyright by me, but it would be nice :-).
- If you use this emodule, especially if you distribute a program which uses this emodule, please send me a mail.
- If you use this emodule, you use it on your own risk!!

Disclaimer

~~~~~

This files are provided "AS IS" without warrenty of any kind, expressed or implied! I'm NOT liable to you for damages or problems, including any general, special, incidental or consequential damages or problems arising out of the use or inhability to use of the files. Including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of a program build with this files.

Or in short words: You use this files on your own risk!
