EVPaths

Version 1.41

Giuseppe Ghibò
`ghibo@galileo.polito.it`

December 10, 1995

1 History

When the firsts TEX related programs were available for the Amiga, each one of them had its own path searching scheme and rules. Mainly, environment variables were used to store a particular path. Many TEX different programs share the same environment variables, but often, these programs use different rules to indicate a path, essentially dued to the fact that the original path routines were ported and adapted from Unix to Amiga by different persons. This situation inevitably led to incompatibilities, so isn't so rare the case that two different TEX programs cannot be used in the same TEX installation. To put an end to the unsatisfactory situation that led to "path anarchy", I decided to write some path routines from scratch. These routines, named "*EVPaths routines*" (Environment Variables Path Routines) are contained in a linkable library, and are very easy to use. The primary target I would obtain is "path standardization", while the second is to override the limit of the current path scheme.

2 Disclaimer/Distribution

3 Features

Here follow the main features of *EVPaths*:

- No limits to the length of the environment variables used to indicate a path.
- Recursively path searching.
- Recursively environment variables searching.

- Commas, semicolons, spaces, tabs, LFs, FFs recognized as path separators.
- Support for path names containing spaces, commas and semicolons.
- Support of the Amiga path notation as well part of the Unix notation.
- Compatibility with any OS from 1.3 to 3.1.
- Linkable with programs which don't make use of any C startup code.

4 Description

4.1 Arbitrary length of environment variables

The environment variable used to specify a path may have any length (according to the memory size specified with the *Alloc_EnvPath()* function). Of course if we set the environment variable using the AmigaDOS command `SetEnv`, the length of the variable is truncated to 255 chars. Using instead an Editor to set the environment variable we may have variables of arbitrary length.

4.2 Recursively path searching

Appending one asterisk `*` or two-asterisks `**` to a path element name, then such name is recursively searched for subdirectories. The single asterisk causes one-level directory searching, while the double asterisk causes the search to be extended into all-levels subdirectories[1]. The recursive scan of subdirectories is executed when the function *Init_EnvPath()* is invoked. Generally this function call is placed at the beginning of the program, so if we append two asterisks `**` to a directory containing a huge number (note 'huge' not 'a few') of non-reorganized subdirectories, this could **slightly** slow down the startup process of the program. Of course isn't a good idea to use path such as:

```
.,dh0:**,dh1:**,dh2:**
```

although this could give an indicative idea of how much time is needed to scan all the subdirectories of the whole hard disk. Now, let's see some examples.

For instance suppose to set the following path:

```
SetEnv TEXINPUTS TeX:texinputs**,TeX:macros*
```

then the directory list could became

[1]The '`**`' notation allows a maximum of 10 level subdirectories (should be enough, but anyway this value can be increased changing the macro `MAX_RECURS_DIR` in the file `EVPaths.c`).

```
TeX:texinputs
TeX:texinputs/one
TeX:texinputs/one/one
TeX:texinputs/one/one/two
TeX:texinputs/one/one/two/three
TeX:texinputs/one/one/two/three/four
        .
        .
        .
TeX:texinputs/two
TeX:texinputs/two/one
TeX:texinputs/two/one/two
        .
        .
        .
TeX:texinputs/macro
TeX:texinputs/macro/one
TeX:texinputs/macro/two
TeX:texinputs/macro/three
```

This mean that when the function *EVP_FileSearch()* or *EVP_Open()* or *EVP_fopen()* is invoked to search a file, then the file is searched into each of the directories above.

One of the most advantage of using the asterisk notation is that we don't have to change the content of an environment variable each time we install a new macro package. Since there are many macro packages for TEX, it's a good idea to keep each of them into his own directory. For instance the LATEX 2ε package contains many sub-tools, and it is a good idea to keep each tool into his own directory[2] rather to put thousands of files into `TeX:texinputs`. For instance with

```
SetEnv TEXINPUTS TeX:texinputs/LaTeX2e*
```

we match

```
TeX:texinputs/LaTeX2e
TeX:texinputs/LaTeX2e/tools
TeX:texinputs/LaTeX2e/graphics
TeX:texinputs/LaTeX2e/babel
```

and so on.

In table 1 are shown some synonymous for the asterisk notation.

Note that the chars `*` and `**` and their synonymous don't specify wildcards; e.g. specifying `TeX:macro/ltx*my` we **don't** include every directory

[2]This is also estabilished by the TWG for the TEX Directory Structure.

Char	Search to
`*`	one-level subdirectories
`**`	all-levels subdirectories
`#?`	one-level subdirectories
`#?>`	all-levels subdirectories
`*>`	all-levels subdirectories

Table 1: Synonymous for `*` and `**`.

matching the string `ltx*my`. Note also that specifying `TeX:texinputs*` or `TeX:texinputs/*` is the same, as well `TeX:texinputs` or `TeX:texinputs/`.

4.3 Recursively environment variable searching

In the path of an enviroment variable we can also specify the name of another enviroment variable. For instance specifying the following path

```
.,TeX:texinputs,$MYPATH,TeX:macros
```

then the contents of the environment variable `MYPATH` will be appended to the directory list. Note that the variable `MYPATH` could contain another environment variable and so on. Up to 5 levels of enviroment variables recursion are allowed[3]. Let's consider the following example:

```
SetEnv VARONE   one,"$"VARTWO
SetEnv VARTWO   two,"$"VARTHREE
SetEnv VARTHREE three,"$"VARFOUR
SetEnv VARFOUR  four,"$"VARFIVE
SetEnv VARFIVE  five,"$"VARSIX
SetEnv VARSIX   six,"$"SEVEN
SetEnv SEVEN    seven
```

then suppose we use the variable `VARONE` for searching. In such case we obtain the following directory list:

```
one
two
three
four
five
```

Since recursive searching of environment variables is limited to 5-levels, the variables `VARSIX` and `VARSEVEN` are ignored. Note that it is necessary to enclose

[3]This value can be increased changing the macro `MAX_RECURS_VAR` in the file `EVPaths.c`.

the `$` between quotes, otherwise the Shell expand the enviroment variable (if it exists). This means that quotes aren't needed if we use an editor to set the environment variable argument. Note that closed loops and duplicated entries are discarded; for instance let's consider

```
SetEnv TEXINPUTS .,TeX:texinputs,"$"MFINPUTS
SetEnv MFINPUTS  .,MF:mfinputs,"$"TEXINPUTS
```

Using `TEXINPUTS`, then the directory list will contain

```
TeX:texinputs
MF:mfinputs
```

while using `MFINPUTS` for file searching, the directory list will became instead

```
MF:mfinputs
TeX:texinputs
```

What to do with a such feature? For instance we could specify a path using **two** environment variables. For example suppose the variable `TEXINPUTS` contains:

```
.,$MYPATH,TeX:texinputs,TeX:LaTeX,TeX:LaTeX2e*
```

We could set the enviroment variable `MYPATH` to `mymacrodir` and then run TEX. Now suppose we would use the macro files of the directory `mymacrodir2`. In such case we have just to change the variable `MYPATH` without to change the whole contents of the variable `TEXINPUTS`. Of course if the variable `MYPATH` is unset, then it is simply ignored.

4.4 Path separators

Each directory in the path may be separated from the other directories using

- Commas
- Semicolons
- Spaces
- Tabs
- Line feeds
- Form feeds

or any combination of them. For example we may edit the variable `TEXINPUTS` as follow

```
        TeX:texinputs,       TeX:macros**
        TeX:latex2e*;    CWeb:macros
        MF:inputs*
        ^L^L
        OldTeX:mymacros
```

and so on.

4.5 Dir names containing path separator chars

If a directory name contains spaces, commas or semicolons we have to enclose such name between quotes. For instance

```
        "The LaTeX:", "this,,  ,,,dirs"
        "Another;;;,,;;;one"
```

Note that if a quote is left open (unmatched quote) then the path element is taken as a directory name until a line feed or form feed is encountered.

4.6 Current directory

To include the current directory into a path we may use one of the following symbols

```
        .       ""      "."
```

4.7 Unix-like path notation

The Unix-like path notation is also supported. For instance the use of

```
./mydir
```

in the path, indicates that the directory `mydir` in the current directory must be placed in the directory list. Instead with

```
../mydir
```

the directory `mydir` of the parent directory is used. Note that the Unix absolute path notation (e.g. `/tex/texinputs/`) isn't supported because it conflicts with the Amiga parent dir notation.

4.8 Default path

If the path specified in the environment variable superseeds the default path, we may use the character `?` to place the default path at an arbitrary position in the directory list. For instance suppose to have the following default path

```
default_path = "MF:inputs,MF:ams/fonts/symbols"
```

and the environment variable

```
SetEnv MFINPUTS .,MF:mfinputs,?,CTAN:mfinputs
```

then the result directory list will be:

```
‘’                        (null string is current dir)
‘MF:mfinputs’             (from MFINPUTS)
‘MF:inputs’               (from default path)
‘MF:ams/fonts/symbols’    (from default path)
‘CTAN:mfinputs’           (from MFINPUTS)
```

4.9 Ignoring duplicated entries

EVPaths ignores duplicated entries. For instance let's consider the following path

```
.,".","",TeX:inputs,MF:inputs,TeX:inputs,MF:inputs"
```

In such case the directory list contains only the following entries

```
‘’               (null string is current dir)
‘TeX:inputs’
‘MF:inputs’
```

5 Programming

Here follow a brief description on how to use *EVPaths* in our C programs.

Firstly simply add the line

```
#include <evpaths.h>
```

in your program.

Then we have to define pointers to the structure `EnvVarPath`. We need one pointer for each environment variable. For instance with

```
struct EnvVarPath *var_one, *var_two;
```

we plan to use two environment variables. Then we have to allocate the memory for each `EnvVarPath` structure. For this purpose we have to use the function *Alloc_EnvVarPath()*. Its synopsis is

```
STRPTR varname;
LONG size;

var_one = Alloc_EnvVarPath(varname, size);
```

where `varname` is the pointer to the name of the environment variable and `size` is the size of the memory needed to store the directory list. Note that each directory name stored in the directory list will consume (*strlen(dirname)* + 5) bytes. For example with

```
var_one = Alloc_EnvVarPath("TEXINPUTS", 4096L);
var_two = Alloc_EnvVarPath("MFINPUTS", 2048L);
```

we alloc 4096 bytes for the path specified in the environment variable `TEXINPUTS` and 2048 bytes for the path specified by `MFINPUTS`. Note that this size doesn't directly depend by the length of the environment variable. For instance an environment variable containing the string `Work:**` is only 7 bytes long but probably the volume `Work:` contains many other directories and subdirectories, so a large value for `size` should be provided. After to have obtained a valid pointer from the function *Alloc_EnvVarPath()* the structure must be initialized by the function *Init_EnvVarPath()*. This is the function which expand and stores the path specified in the environment variable. Its template is

```
Init_EnvVarPath(struct EnvVarPath *,
                APTR defpath,
                LONG mode);
```

For instance

```
STRPTR def_path_str   = ".,TeX:inputs*";
STRPTR def_path_arr[] = { ".","TeX:inputs*", NULL };

Init_EnvVarPath(var_one, def_path_str, ENVPATH_DEFSTR);
Init_EnvVarPath(var_two, def_path_arr, ENVPATH_DEFARR);
```

The constant `ENVPATH_DEFARR` must be used to indicate that the default path is a pointer to an array of strings, while the constant `ENVPATH_DEFSTR` must be used, instead, if the passed default path is a `STRPTR` pointer. If we don't have a default path we have to specify `NULL` as argument, e.g.

```
Init_EnvVarPath(var_one, NULL, NULL);
```

When the function *Init_EnvVarPath()* is called, the buffer allocated for the `EnvVarPath` structure is filled with entries either from the environment variable and/or from the default path. Path elements containing `*` or `**` or `?` or names of other environment variables are expanded (i.e. the scan of the directory tree is performed).

Note that the default path is overridden by the contents of the environment variable. It is used just in case the specified environment variable doesn't exists. It is also possible to prepend or append the default path to the one read from the environment variable. To do this simply **OR** the third argument of the *Init_EnvVarPath()* function with the constants `ENVPATH_PREPEND_PATH` or `ENVPATH_APPEND_PATH`. For example

```
        Init_EnvVarPath(var_one,
                        def_path_str,
                        ENVPATH_DEFSTR | ENVPATH_PREPEND_PATH);
```

In this case the default path is used for searching, **before** any other path element given in the environment variable. Instead with

```
        Init_EnvVarPath(var_one,
                        def_path_str,
                        ENVPATH_DEFSTR | ENVPATH_APPEND_PATH);
```

the default path is searched **after** the path given in the environment variable. The example below can make this clear. Suppose the directory `TeX:inputs` contains only two subdirectories (e.g. `one` and `two`). Now suppose to set the variable `TEXINPUTS` with the command

```
            SetEnv TEXINPUTS .,TeX:texinputs,TeX:macros
```

Suppose also to have the following default path

```
                  default_path = ".,TeX:inputs*";
```

With

```
        Init_EnvVar(var_one,
                    default_path,
                    ENVPATH_DEFSTR|ENVPATH_APPEND_PATH);
```

the directory list became

```
        ‘’                  (null dir is current dir)
        ‘TeX:texinputs’     (from the env var TEXINPUTS)
        ‘TeX:macros’        (from the env var TEXINPUTS)
        ‘TeX:inputs’        (from the default path)
        ‘TeX:inputs/one’    (from the default path)
        ‘TeX:inputs/two’    (from the default path)
```

Note how the current dir entry (`.`), present either in the default path and in the environment variable, is considered only once.

There is another option we can specify (together with the others explained above) in the third argument of the function *Init_EnvVarPath()*. **OR**ing with `ENVPATH_CURRENTDIR_FIRST` then the current directory is used to search files **before** any other path specified in the default path and/or in the environment variable.

Note that the call to the *Init_EnvVarPath()* may be placed anywhere in the program, but is a good practice to place it at the beginning of the program. The important is that the `EnvVarPath` structure is initialized before calling any of the functions listed below.

Once the pointer to the `EnvVarPath` structure has been initialized by the function *Init_EnvVarPath()* we may inquire the field `evp->status` to know the status of the buffer. The possibles values are listed in the file `EVPaths.h` (i.e. EVPATHS_BUFFER_EMPTY, ENVPATH_BUFFER_FULL, . . .).

5.1 Searching a file

5.1.1 EVP_FileSearch()

After the `EnvVarPath` structure pointer has been initialized as explained above, we can search a file using the function *EVP_FileSearch()*. Its template is:

```
STRPTR EVP_Filesearch(STRPTR filename,
                      struct EnvVarPath *evp,
                      UBYTE *buffer,
                      LONG size);
```

where `filename` is a pointer to the name of the file to search, `buffer` of size `size` is a buffer used to store the name of the file found. If the file is found then the returned value is a pointer to `buffer` and `buffer` will contain the full name of the file, otherwise a NULL pointer is returned. The example below may clarify the correct use of the function *EVP_FileSearch()*

```
#include <evpaths.h>

#define SIZE    256L
#define EVPSIZE 4096L

struct EnvVarPath *mf_var;
char *default_path = ".,MF:mfinputs*";

void main()
{
        STRPTR s;
        UBYTE bufname[SIZE];

        evp = Alloc_EnvVarPath("MFINPUTS", EVPSIZE);
        Init_EnvVarPath(mf_var,
                        default_path,
                        ENVPATH_DEFSTR);

        if (s = EVP_FileSearch("cmr10.mf",
                               mf_var, bufname, SIZE))
        {
                printf("File found: %s\n", s);
        }
        else
        {
                printf("File not found!\n");
        }

        Free_EnvVarPath(mf_var);
}
```

In this example the file `cmr10.mf` is searched using the path provided in the environment variable `MFINPUTS`.

5.1.2 EVP_Open()

The function *EVP_Open()* combines the features of the function *EVP_FileSearch()* and of the AmigaDOS function *Open()* in one function. Its template is:

```
        BPTR EVP_Open(STRPTR filename,
                      struct EnvVarPath *evp,
                      UBYTE *buffer,
                      LONG size,
                      LONG mode);
```

The arguments `filename`, `evp`, `buffer` and `size` are the same as in the function *EVP_FileSearch()*; `mode` is the opening mode as in the AmigaDOS function *Open()* (i.e. `MODE_OLDFILE` or `MODE_NEWFILE`). This function returns the `BPTR` of the opened file exactly in the same way the function *Open()* does, i.e. if the file is found in the provided path it returns a valid `BPTR`, otherwise `NULL` is returned. The full name of the found file is stored in the provided buffer `buffer`. If we don't need to know the full name of the file found we have just to use `NULL` as argument for `buffer` and `size`, e.g.

```
   fh = EVP_Open("cmr10.mf", mf_var, NULL, NULL, MODE_OLDFILE);
```

For instance using this function in the example above we obtain

```
                .
                .
                .

                BPTR fh;

                if (fh = EVP_Open("cmr10.mf",
                                  mf_var, bufname, MODE_OLDFILE))
                        printf("File found and opened!\n");
                else
                        printf("File not found!\n");

                Free_EnvVarPath(mf_var);
        }
```

While the behaviour of *EVP_Open()* with `MODE_OLDFILE` is rather clear, what we can do with `MODE_NEWFILE` isn't so immediate. Suppose to set the environment variable `TEXCONFIG` as follows

```
SetEnv TEXCONFIG TeX:config,.
```

Now suppose we to save a config file. We may use in such case

```
        fh = EVP_Open("mf.cmf",
                       mfcfg_var, buffer, size, MODE_NEWFILE);
```

If the file `TeX:config/mf.cmf` already exists but it's protected against writing or deleting, or the directory `TeX:config` doesn't exists, then the next path element (e.g. current directory in the example) will be used to save the file `mf.cmf`.

5.1.3 EVP_fopen()

The function *EVP_fopen()* is similar to the function *EVP_Open()* except it returns a `FILE *` instead of `BPTR`. The argument `mode` of this function is the same as in the C function *fopen()*. Template:

```
        FILE *EVP_Open(STRPTR filename,
                       struct EnvVarPath *evp,
                       UBYTE *buffer,
                       LONG size,
                       char *mode);
```

Note to use this function you need to use the library `EVPaths_no.lib` instead of the library `EVPaths.lib`.

5.2 Freeing allocated vars

Before the program exits all the allocated memory must be freed. For this purpose the function `Free_EnvVarPath(env_var)` must be called before the program exits. This is a care stuff, otherwise the program will not release all the allocated memory. Note that if a CTRL-C interrupts the program before it reaches the point of Free_EnvVarPath(), an interrupt handler should be provided (this could be done using the function *signal()*). Alternatively we can use the library `EVPaths_no.lib`, which does not requires any interrupt handler because it uses the function *malloc()* instead of *AllocMem()* to alloc the memory: However remember that *EVPaths_no.lib* cannot be used if the program doesn't make use of any C standard startup code.

5.3 Examples

The files `TeXWhereis.c`, `TeXWhereIs2.c` and `TeXSaveString.c` contain some examples on how to use *EVPaths*.

6 Programs supporting *EVPaths*

Currently the following Amiga programs support *EVPaths*

- METAFONT 2.71, METAFONT 2.718.
- METAPOST 0.63.
- PasTEX $\geq$ 1.4.
- *MakeIndex* 2.13.
- dvips 5.58 $\geq$ revision 1.
- AmiWeb2c 6.1 $\geq$ release 1.0.

7 Acknowledgements

The author wishes to thanks Andreas Scherer and Georg Heßmann for their significative suggestions and collaboration and for having intensely tested *EVPaths*.

8 Misc

For any questions, suggestions, comments, bug report or enhancement requests, please feel free to e-mail me to `ghibo@galileo.polito.it`.