The 2.0 Dos.library Path Name Handling Functions

By Ewout Walraven

When processing file names, it is often necessary to extract only a file name from a path or to generate an absolute path to a file. The release 2.0 *dos.library* contains several routines designed to make this easier.

The FilePart() function takes a pointer to a file path and returns a pointer to the last part of the string (the part of the string that follows the last separator character, ‘‘/’’). The last part of the string is normally a file or directory name.

```
UBYTE *FilePart(UBYTE *mypath);
```

In case mypath (from the prototype above) consists only of a file or directory name (like *startup-sequence*, *s*, or *libs*), FilePart() returns a pointer to the start of the string (a.k.a. the same value as mypath).

The PathPart() function returns a pointer to the last character in the path string that doesn’t include the file name (usually the separator character ‘‘/’’):

```
UBYTE *PathPart(UBYTE *mypath);
```

If passed a pointer to the path string ‘‘sys:s/startup-sequence’’, PathPart() will return a pointer to ‘‘/startup-sequence’’. The application can then put a \0 where PathPart() points so that the original string has been shortened to a path part string. In case mypath (from the above prototype) is only a file name, PathPart() returns a pointer to this file name (the same value as mypath). This can confuse an application if it blindly expects PathPart() to return a path. An application should make sure the pointer PathPart() returns is not the same pointer the application passed to the function.

Both FilePart() and PathPart() consider the initial colon of an absolute path name (a path starting with a volume/device/assign name) to be a special separator character, but PathPart() will not include the colon in the string it returns a pointer to. For example, if passed the string ‘‘ram:tmpfile’’, PathPart() will return a pointer to the string ‘‘tmpfile’’.

Neither FilePart() nor PathPart() check the syntax of the path string passed to them, so if your application passes an invalid path to them, they will pass back equally invalid path fragments. Also, they do not process any of the wildcard tokens, treating them as normal characters.

The following chart summarizes the possible results of the FilePart() and PathPart() functions. It assumes that the path supplied to the functions is valid.

wholepath is the path passed to FilePart() and PathPart(),
pathpart is result of PathPart(),
and filepart is the result of FilePart().

1. If (wholepath != pathpart != filepart), then filepart points to the the file or directory name in wholepath and pathpart points to the '/' before the file or directory name in wholepath. For example:

    wholepath: ‘‘ram:t/tmpfile’’
    Filepart: ‘‘tmpfile’’
    Pathpart: ‘‘/tmpfile’’

    wholepath: ‘‘//t/tmpfile’’
    Filepart: ‘‘tmpfile’’
    Pathpart: ‘‘/tmpfile’’

2. If (wholepath != pathpart == filepart), then filepart and pathpart point to a file or a directory name preceded by a volume name or series of separator characters. For example:

    wholepath: ‘‘ram:tmpfile’’
    Filepart: ‘‘tmpfile’’
    Pathpart: ‘‘tmpfile’’

    wholepath: ‘‘//tmpfile’’
    Filepart: ‘‘tmpfile’’
    Pathpart: ‘‘tmpfile’’

3. If (wholepath == pathpart == filepart) then pathpart points to either a file or a directory name. This name is not preceded by a volume name.

    wholepath: ‘‘tmpfile’’
    Filepart: ‘‘tmpfile’’
    Pathpart: ‘‘tmpfile’’

4. If (pathpart[0] == 0), then path points to either "" (current directory), a series of separators (‘‘/’’, ‘‘//’’, ‘‘:’’, etc.), or a volume/device name.

    wholepath: ‘‘////’’
    Filepart: ‘‘’’
    Pathpart: ‘‘’’

    wholepath: ‘‘ram:’’
    Filepart: ‘‘’’
    Pathpart: ‘‘’’

The *dos.library* AddPart() function lets you append a file or directory name to the end of a (relative) path, inserting any necessary separator characters:

```
BOOL AddPart(UBYTE *mypathname, UBYTE *myfilename, ULONG mysize);
```

AddPart() will add myfilename to the end of mypathname. The mysize argument indicates how large the mypathname buffer is. In case this buffer is too small, AppPart() returns FALSE and makes no changes to the buffer. If myfilename is an absolute path, it will replace the current contents of the mypathname buffer. If myfilename starts with a colon, AddPart() will generate a new path relative to the root of the volume/device in mypathname.

These functions are used in subsequent examples. The *Part.c* example lets you view the results of these functions. Taking a path and a file name as arguments, *Path.c* interprets the path, removes the file name (if part of) from the path and replaces the path's file name with the file name supplied in the argument list.

Another *dos.library* function called SplitName() makes it possible to extract individual volume/device, directory, or file names (referred to here as path components) from a path string:

```
WORD = SplitName(UBYTE *mypathname, UBYTE separatorchar, UBYTE *buffer, WORD oldposition,
                 LONG buffersize);
```

SplitName() searches through the string mypathname for the separator character (separatorchar from the above prototype) starting at oldposition. As it is stepping through mypathname, SplitName() copies the characters it encounters from mypathname to the buffer, terminating the resulting string with a NULL. It *does not* copy the separator character into the buffer. If it finds the separator character, SplitName() returns the position of the character that follows the separator character. This position can be used as the oldposition argument in subsequent calls to SplitName() to extract other path components from the same path. If SplitName() does not find another separator character, it will return -1, although it will still copy characters from mypathname to buffer. For example, if SplitName() was called with the following arguments:

```
mypathname = "ram:env/sys/win.pat"
separatorchar = '/'
oldposition = 8
buffersize = 10
```

buffer would contain the NULL terminated string ‘‘sys’’ and SplitName() would return a value of 12. If SplitName() was called again placing the return value of 12 into oldposition, buffer would contain the NULL terminated string ‘‘win.pat’’ and SplitName() would return a value of -1, as there are no separator characters in mypathname beyond the eleventh character.

If SplitName() runs out of room in the buffer, it will copy as much of the characters as it can fit (buffersize - 1) and will write a NULL into the last position in the buffer (actually, if a path component is followed by a separator character and the 2.04 version SplitName() runs out of room in the buffer, SplitName() will only copy (buffersize - 2) characters into the buffer. This may be rectified in the future). Also, if SplitName() runs out of room in the buffer *and* it finds another separator character, the position it returns will *not* be of the character that follows the separator character. Instead, SplitName() will return the position of the actual separator character.

The *Split.c* example shows the behavior of this function. The example takes two arguments, the path name to split up and the size of the destination buffer SplitName() should use for the path components.