CoreNGS Filesystem Library for GameMaker Studio 2.3.1+

string directory_get_current_working()

If successful, the function returns a null-terminated string containing an absolute pathname that is the current working directory of the calling process, while including a trailing slash. If the function fails, an empty string is returned. The current working directory is the directory, associated with the process, that is used as the starting location in pathname resolution for relative paths.

To change the current working directory, call directory_set_current_working(). Since the working directory can be changed at any given time by a function or library call, there are many possible cases where the current working directory shall not equal the same absolute pathname as the path to the executable as returned by executable_get_directory().

real directory_set_current_working(string dname)

If successful, the function changes the current working directory of the calling process to the directory specified in dname and shall return true. If the function fails, the current working directory is not changed and false is returned. To retrieve the current working directory, call directory_get_current_working(). Environment variables found in dname in the form of ${VARIABLE} are automatically expanded.

string directory_get_temporary_path()

If successful, the function returns a null-terminated string containing a directory suitable for temporary files, while including a trailing slash. If the function fails, it shall return an empty string.

string executable_get_directory()

If successful, the function returns a null-terminated string containing a directory which holds the executable file of the calling process, while including a trailing slash. If the function fails, it shall return an empty string.

string executable_get_filename()

If successful, the function returns a null-terminated string containing the filename of the executable file of the calling process. If the function fails, it shall return an empty string.

string executable_get_pathname()

If successful, the function returns a null-terminated string containing the absolute pathname, (both the directory and filename), of the executable file of the calling process. If the function fails, it shall return an empty string.

string file_bin_pathname(real fd)

If successful, the function returns a null-terminated string containing absolute pathname of the file or directory associated with the specified file descriptor in fd. If the function fails, it shall return an empty string. On Unix-like systems, there can be multiple possible absolute pathnames for a given file descriptor due to multiple hardlinks. The first hardlink found that matches the file descriptor shall be the absolute pathname returned. Also note not all file descriptors point to a file which can be represented with a pathname on disk, such as pipes.

string filename_absolute(string fname)

If successful, this function shall resolve all symbolic links and expand all relative path symbols passed to fname, and shall return a null-terminated string containing the absolute path equivalent of fname. If fname points to an existing directory, a single trailing slash is appended to the return value. If fname does not point to an existing regular file or directory, an empty string is returned and the function fails. If the function fails for any other reason, it shall return an empty string. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

string filename_canonical(string fname)

If successful, this function shall resolve all symbolic links and expand all relative path symbols passed to fname, and shall return a null-terminated string containing the absolute path equivalent of fname. If fname points to an existing directory, a single trailing slash is appended to the return value. The function shall succeed even if the absolute path resolved does not point to an existing regular file or directory, in which case the portions of the path which do not exist are appended after the portion of the absolute path that does exist. If the function fails, it shall return an empty string. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_exists(string fname)

returns true if fname points to an existing regular file and not a directory, otherwise false is returned. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_delete(string fname)

attempts to delete a regular file, (not a directory), pointed to by fname, and returns true when the file was successfully deleted, otherwise false is returned. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_rename(string oldname, string newname)

attempts to rename a regular file, (not a directory), pointed to in oldname to the new pathname found in newname. Returns true on success, false on failure. Environment variables found in oldname and newname in the form of ${VARIABLE} are automatically expanded.

real file_copy(string fname, string newname)

attempts to copy a regular file, (not a directory), pointed to in fname to the new pathname found in newname. Returns true on success, false on failure. Environment variables found in fname and newname in the form of ${VARIABLE} are automatically expanded.

real file_size(string fname)

retrieves the file size, (in bytes), of the regular file, (not a directory), pointed to by fname. Returns zero if the regular file was not found or some other error occurred with reading the file size. To read the file size of an opened file descriptor, call file_bin_size(). Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real directory_exists(string dname)

returns true if dname points to an existing directory, (not a regular file), otherwise false is returned. Environment variables found in dname in the form of ${VARIABLE} are automatically expanded.

real directory_create(string dname)

attempts to create one or more new directory entries recursively, pointed to by the path in dname, and returns true when the directory, (or directories), were successfully created, otherwise false is returned. Environment variables found in dname in the form of ${VARIABLE} are automatically expanded.

real directory_destroy(string dname)

attempts to destroy a directory along with all of its contents recursively based on the directory dname points to. Returns true if the operation completed successfully, otherwise false is returned. Environment variables found in dname in the form of ${VARIABLE} are automatically expanded.

real directory_rename(string oldname, string newname)

attempts to rename a directory, (not a regular file), pointed to in oldname to the new pathname found in newname. Returns true on success, false on failure. Environment variables found in oldname and newname in the form of ${VARIABLE} are automatically expanded.

real directory_copy(string dname, string newname)

attempts to copy a directory, (not a regular file), pointed to in dname to the new pathname found in newname. Returns true on success, false on failure. Environment variables found in dname and newname in the form of ${VARIABLE} are automatically expanded.

real directory_size(string dname)

retrieves the file size, (in bytes), of the directory, (not a regular file), pointed to by dname. Returns zero if the directory was not found or some other error occurred with reading the file size. To read the file size of an opened file descriptor, call file_bin_size(). Environment variables found in dname in the form of ${VARIABLE} are automatically expanded.

real directory_contents_get_order()

returns the enumeration value which is associated with the current ordering to be assigned with directory iteration. To see the full list of possible enumeration values that may be returned by this function, refer to the possible values that may be passed to directory_contents_set_order(). The default return value of directory_contents_get_order() is zero, (which corresponds to alphabetical order).

real directory_contents_set_order(real order)

the function sets the order of the file list returned by directory_contents_first() and directory_contents_next(). This function must be called before a call to directory_contents_first() so that the changes in order may take effect. For the argument order you may pass one of the following enumeration values to determine this order:

```
// Directory Contents
#define DC_ATOZ  0 // Alphabetical Order
#define DC_ZTOA  1 // Reverse Alphabetical Order
#define DC_AOTON 2 // Date Accessed Ordered Old to New
#define DC_ANTOO 3 // Date Accessed Ordered New to Old
#define DC_MOTON 4 // Date Modified Ordered Old to New
#define DC_MNTOO 5 // Date Modified Ordered New to Old
#define DC_COTON 6 // Date Created Ordered Old to New
#define DC_CNTOO 7 // Date Created Ordered New to Old
#define DC_RAND  8 // Random Order
```

real directory_contents_get_cntfiles()

The function returns the number of counted files and/or directories that were iterated through from a previous call to directory_contents_first(). Note a call to directory_contents_first() will store the entire array of files and directories that match the given criteria in advance to predetermine the return value of later calls to directory_contents_next() thereafter, and directory_contents_get_cntfiles() will return the entire number of files and directories that were iterated through by the call to directory_contents_first() internally. directory_contents_get_cntfiles() will return the same result regardless of whether it is called before or after any number of calls to directory_contents_next().

The thing to remember is it needs to be called after at least one call to directory_contents_first() and it will already return the value you need at that point. directory_contents_get_cntfiles() does not return the value of the number of files or directories that match the criteria specified by directory_contents_first(). It returns every single file and/or directory in the search provided by directory_contents_first(), regardless of the amount of files and/or directories which matched the criteria. The default return value is one. If more than one file and/or directory is searched through, the return value is increased to be that total number of files and/or directories which were iterated through in the search.

real directory_contents_get_maxfiles()

The function returns the maximum number of files and/or directories to search through in a call to directory_contents_first() before ending the search that is looking for the specified criteria in the arguments passed to directory_contents_first(). Returns zero by default. The return value of this function may be set by directory_contents_set_maxfiles().

real directory_contents_set_maxfiles(real maxfiles)

The function sets the maximum number of files to search through for testing against whether they match the criteria of the arguments passed to directory_contents_first(). Pass zero to the maxfiles argument to represent no maximum value and allow for there to be zero limitations in place. Using no limitation can cause a search to take a very long time when searching directories with many files or directories in them, so it is preferred to set a limitation of some kind in applications that shall block the main thread while the search is being performed by directory_contents_first().

Otherwise, the user might mistake the search for the program not responding or get tired of waiting for it to finish. The directory_contents_first() and directory_contents_next() functions are not thread-safe and should only be called from a single-thread at a time. In a multi-threaded application, you may provide a progress indicator using an implementation of your choosing but you it is wise to allow for a means to cancel the search and force the thread to quit its execution while also being asynchronous and non-blocking when possible.

Test for equality, (and not less-than, or less-than-or-equal-to), between directory_contents_get_cntfiles() and directory_contents_get_maxfiles(), and, if the two are equal, let this be a means to cancel the operation of looping through following calls to directory_contents_next(), which will apply the proper limitations set by directory_contents_set_maxfiles() and prevent undesirable hanging of the main thread when running in a single-threaded application or program. This will helps improve user experience where such behavior in an application is at all acceptable. Otherwise it is in most cases better to write a multi-threaded application. This function has no return value.

string directory_contents_first(string dname, string pattern, real includedirs, real recursive)

If successful, the function shall return the first regular file or directory found in the pathname pointed to in dname, which abides by the specified pattern. The pattern is semicolon separated if more than one file type glob is to be found in the search. A string that is acceptable for the pattern argument could look like, for example: “*.png;*.gif;*.PNG;*.GIF”. Do note file extensions used in pattern are case sensitive so it is best to include both an all-caps and all-lowercase versions of each file extension used. If includedirs is true, directories shall be included in the search, and not just regular files. If recursive is true, the search shall retrieve directory contents from all sub-directories, and not just the initial directory specified in dname. Call directory_contents_next() after this function to get the rest of the files matching the desired criteria in the search.

To change the order of the files returned by directory_contents_first() and directory_contents_next(), call directory_contents_set_order(). If the function fails or there are no files matching the given criteria, an empty string is returned. Alphabetical order is the default. The returned string shall be null-terminated. Remember to free memory after a call to this function with directory_contents_close() when there is no longer a need to collect returned values from directory_contents_first() or directory_contents_next(). directory_contents_first() shall free memory with an automatic call to directory_contents_close() before attempting to write to new memory. Environment variables found in dname in the form of ${VARIABLE} are automatically expanded.

string directory_contents_next()

If successful, the function returns the next file in the list of files searched for in a previous call to directory_contents_first(). The returned string shall be null-terminated. directory_contents_next() should be called in a while or for loop that breaks when no more files are found in the search, that is, when directory_contents_next() returns an empty string to indicate failure. Store the returned values of each call to this function copied into an array for later use. Note in the case of a race condition, the files may no longer exist at any point from the time they were found in the search to the time they were needed for use, so it is good to always check for regular file's, (or directory's), existence before trying to make use of those files or directories.

real directory_contents_close()

Frees memory from a previous function call to directory_contents_first(), which shall cause new calls to directory_contents_next() to return an empty string consequently, unless the storage is filled again by another call to directory_contents_first(). The function is safe to call even when no memory is in need of being freed.

string environment_get_variable(string name)

The function shall search the environment of the calling process for the environment variable name if it exists and return a pointer to the value of the environment variable. If the specified environment variable cannot be found, an empty string shall be returned.

real environment_set_variable(string name, string value)

The function shall update or add a variable in the environment of the calling process. The name argument points to a string containing the name of an environment variable to be added or altered. The environment variable shall be set to the value to which the value argument points. The function shall fail if name points to a string which contains an '=' character. The function returns true on success, or false on failure.

real environment_unset_variable(string name)

The function shall remove an environment variable from the environment of the calling process. The name argument points to a string, which is the name of the variable to be removed. The named argument shall not contain an '=' character. If the named variable does not exist in the current environment, the environment shall be unchanged and the function is considered to have completed successfully. The function returns true on success, or false on failure.

string environment_expand_variables(string str)

The function expands environment variables found in the string pointed to in the str argument, which are in the form of ${VARIABLE}. In Windows Command Prompt, this is the equivalent of entering %VARIABLE%, and on a Unix-like shell, this is the same as entering $VARIABLE. If the environment variable is not found, ${VARIABLE} is substituted with an empty string. VARIABLE in ${VARIABLE} can be whatever string you would like which matches a variable name in the calling process's environment block. The variable name shall not contain an '=' character.

real file_datetime_accessed_year(string fname)

If successful, the function returns the year accessed timestamp of the given filename pointed to in fname. Possible values range from 1900 to the present. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_accessed_month(string fname)

If successful, the function returns the month accessed timestamp of the given filename pointed to in fname. Possible values range from 1 to 12. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_accessed_day(string fname)

If successful, the function returns the day of the month accessed timestamp of the given filename pointed to in fname. Possible values range from 1 to 31. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_accessed_hour(string fname)

If successful, the function returns the hour accessed timestamp of the given filename pointed to in fname. Possible values range from 0 to 23. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_accessed_minute(string fname)

If successful, the function returns the minute accessed timestamp of the given filename pointed to in fname. Possible values range from 0 to 59. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_accessed_second(string fname)

If successful, the function returns the second accessed timestamp of the given filename pointed to in fname. Possible values range from 0 to 61. It generally is 0 to 59. The extra range is to accommodate for leap seconds in certain systems. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_modified_year(string fname)

If successful, the function returns the year modified timestamp of the given filename pointed to in fname. Possible values range from 1900 to the present. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_modified_month(string fname)

If successful, the function returns the month modified timestamp of the given filename pointed to in fname. Possible values range from 1 to 12. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_modified_day(string fname)

If successful, the function returns the day of the month modified timestamp of the given filename pointed to in fname. Possible values range from 1 to 31. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_modified_hour(string fname)

If successful, the function returns the hour modified timestamp of the given filename pointed to in fname. Possible values range from 0 to 23. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_modified_minute(string fname)

If successful, the function returns the minute modified timestamp of the given filename pointed to in fname. Possible values range from 0 to 59. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_modified_second(string fname)

If successful, the function returns the second modified timestamp of the given filename pointed to in fname. Possible values range from 0 to 61. It generally is 0 to 59. The extra range is to accommodate for leap seconds in certain systems. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_created_year(string fname)

If successful, the function returns the year created timestamp of the given filename pointed to in fname. Possible values range from 1900 to the present. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_created_month(string fname)

If successful, the function returns the month created timestamp of the given filename pointed to in fname. Possible values range from 1 to 12. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_created_day(string fname)

If successful, the function returns the day of the month created timestamp of the given filename pointed to in fname. Possible values range from 1 to 31. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_created_hour(string fname)

If successful, the function returns the hour created timestamp of the given filename pointed to in fname. Possible values range from 0 to 23. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_created_minute(string fname)

If successful, the function returns the minute created timestamp of the given filename pointed to in fname. Possible values range from 0 to 59. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_datetime_created_second(string fname)

If successful, the function returns the second created timestamp of the given filename pointed to in fname. Possible values range from 0 to 61. It generally is 0 to 59. The extra range is to accommodate for leap seconds in certain systems. Returns -1 on failure. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_bin_datetime_accessed_year(real fd)

If successful, the function returns the year accessed timestamp of the given file descriptor found in fd. Possible values range from 1900 to the present. Returns -1 on failure.

real file_bin_datetime_accessed_month(real fd)

If successful, the function returns the month accessed timestamp of the given file descriptor found in fd. Possible values range from 1 to 12. Returns -1 on failure.

real file_bin_datetime_accessed_day(real fd)

If successful, the function returns the day of the month accessed timestamp of the given file descriptor found in fd. Possible values range from 1 to 31. Returns -1 on failure.

real file_bin_datetime_accessed_hour(real fd)

If successful, the function returns the hour accessed timestamp of the given file descriptor found in fd. Possible values range from 0 to 23. Returns -1 on failure.

real file_bin_datetime_accessed_minute(real fd)

If successful, the function returns the minute accessed timestamp of the given file descriptor found in fd. Possible values range from 0 to 59. Returns -1 on failure.

real file_bin_datetime_accessed_second(real fd)

If successful, the function returns the second accessed timestamp of the given file descriptor found in fd. Possible values range from 0 to 61. It generally is 0 to 59. The extra range is to accommodate for leap seconds in certain systems. Returns -1 on failure.

real file_bin_datetime_modified_year(real fd)

If successful, the function returns the year modified timestamp of the given file descriptor found in fd. Possible values range from 1900 to the present. Returns -1 on failure.

real file_bin_datetime_modified_month(real fd)

If successful, the function returns the month modified timestamp of the given file descriptor found in fd. Possible values range from 1 to 12. Returns -1 on failure.

real file_bin_datetime_modified_day(real fd)

If successful, the function returns the day of the month modified timestamp of the given file descriptor found in fd. Possible values range from 1 to 31. Returns -1 on failure.

real file_bin_datetime_modified_hour(real fd)

If successful, the function returns the hour modified timestamp of the given file descriptor found in fd. Possible values range from 0 to 23. Returns -1 on failure.

real file_bin_datetime_modified_minute(real fd)

If successful, the function returns the minute modified timestamp of the given file descriptor found in fd. Possible values range from 0 to 59. Returns -1 on failure.

real file_bin_datetime_modified_second(real fd)

If successful, the function returns the second modified timestamp of the given file descriptor found in fd. Possible values range from 0 to 61. It generally is 0 to 59. The extra range is to accommodate for leap seconds in certain systems. Returns -1 on failure.

real file_bin_datetime_created_year(real fd)

If successful, the function returns the year created timestamp of the given file descriptor found in fd. Possible values range from 1900 to the present. Returns -1 on failure.

real file_bin_datetime_created_month(real fd)

If successful, the function returns the month created timestamp of the given file descriptor found in fd. Possible values range from 1 to 12. Returns -1 on failure.

real file_bin_datetime_created_day(real fd)

If successful, the function returns the day of the month created timestamp of the given file descriptor found in fd. Possible values range from 1 to 31. Returns -1 on failure.

real file_bin_datetime_created_hour(real fd)

If successful, the function returns the hour created timestamp of the given file descriptor found in fd. Possible values range from 0 to 23. Returns -1 on failure.

real file_bin_datetime_created_minute(real fd)

If successful, the function returns the minute created timestamp of the given file descriptor found in fd. Possible values range from 0 to 59. Returns -1 on failure.

real file_bin_datetime_created_second(real fd)

If successful, the function returns the second created timestamp of the given file descriptor found in fd. Possible values range from 0 to 61. It generally is 0 to 59. The extra range is to accommodate for leap seconds in certain systems. Returns -1 on failure.

real file_bin_open(string fname, real mode)

If successful, the regular file, (not a directory), pointed to in fname is opened for reading, writing, and/or appending as specified by the mode enumeration argument, and the function returns the file descriptor to be associated with the opened file. The file descriptor may then be passed to other functions which call for one, whether a file_bin_*() function, a file_text_*() function, or any other API that can make use of a valid file descriptor. On failure, -1 is returned. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded. The mode enumeration argument may be one of the following values:

```
// File Descriptors
#define FD_RDONLY 0 // Opened as Read-Only
#define FD_WRONLY 1 // Opened as Write-Only
#define FD_RDWR 2 // Reading and Writing
#define FD_APPEND 3 // Opened for Appending
#define FD_RDAP 4 // Reading and Appending
```

real file_bin_rewrite(real fd)

If successful, the function destroys all contents of the given file descriptor pointed to in fd, and resets the seek position of the file descriptor to zero, leaving the file with empty contents, which is useful for rewriting a file from scratch with completely new or different contents. Returns zero on success, and -1 on failure.

real file_bin_size(real fd)

If successful, the function returns the file size, (in bytes), of an opened file descriptor, or zero on failure. To get the file size, (in bytes), of a regular file based on its pathname, call file_size(). To get the file size, (in bytes), of a directory based on its pathname, call directory_size().

real file_bin_position(real fd)

If successful, the function returns the current seek position, relative to the beginning of the opened file descriptor pointed to by fd, which is a seek position of zero. The seek position should be a value between zero and the file's size, (in bytes). On failure, -1 is returned.

real file_bin_seek(real fd, real pos)

If successful, the function returns the current seek position, relative to the beginning of the opened file descriptor pointed to by fd, which is a seek position of zero. The seek position should be a value between zero and the file's size, (in bytes), which is changed by the value pointed to in pos, relative to the current position. On failure, -1 is returned.

real file_bin_read_byte(real fd)

If successful, the function reads and returns one character byte at the current seek position of an opened file descriptor pointed to by the fd argument. On failure, a null-byte is returned. However, even when there is not a failure case, it is possible to still return a null-byte character, so it is better to test success not against this function, but rather, from file_bin_open(), file_text_open_read(), or a similar function in advance and if any one of those functions return -1, don't attempt to read the file's contents because it can't be opened or read under such conditions. Note the file must also be opened specifically for reading in mind with the specified file descriptor in fd, otherwise the function shall fail.

real file_bin_write_byte(real fd, real byte)

If successful, the function writes one character byte with the value found in the byte argument, at the current seek position of an opened file descriptor, pointed to by the fd argument, while returning zero. Returns -1 on failure. Fails if the specified file descriptor pointed to in fd does not reference a file opened for writing or appending.

real file_bin_close(real fd)

If successful, the function closes an opened file descriptor pointed to by fd, while returning zero on success, and -1 on failure. Always close a file descriptor after you no longer need to read, write, or append data from and/or to it. This function is no different from file_text_close().

real file_text_open_read(string fname)

If successful, the function opens the regular file, (not a directory), found in the fname argument, for reading, and returns the file descriptor to associate with that opened file. On failure, -1 is returned. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_text_open_write(string fname)

If successful, the function opens the regular file, (not a directory), found in the fname argument, for writing, and returns the file descriptor to associate with that opened file. On failure, -1 is returned. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_text_open_append(string fname)

If successful, the function opens the regular file, (not a directory), found in the fname argument, for appending, and returns the file descriptor to associate with that opened file. On failure, -1 is returned. Environment variables found in fname in the form of ${VARIABLE} are automatically expanded.

real file_text_write_real(real fd, real val)

If successful, the function writes to the text file at the current seek position a double-precision numerical value pointed to in val, based on the opened file descriptor referenced in the fd argument, and the number of bytes written are returned. Fails if the file descriptor in fd is not opened for writing or appending. Returns -1 on failure.

real file_text_write_string(real fd, string str)

If successful, the function writes to the text file at the current seek position a string value pointed to in str, based on the opened file descriptor referenced in the fd argument, and the number of bytes written are returned. Fails if the file descriptor in fd is not opened for writing or appending. Returns -1 on failure.

real file_text_writeln(real fd)

If successful, the function writes to the text file a single line-feed character at the current seek position to create a new line, based on the opened file descriptor referenced in the fd argument, and the number of bytes written are returned. Fails if the file descriptor in fd is not opened for writing or appending. Returns -1 on failure.

real file_text_eoln(real fd)

The function returns true if the current seek position of the file descriptor pointed to in fd is at the end of a line or the end of the file, otherwise false.

real file_text_eof(real fd)

The function returns true if the current seek position of the file descriptor pointed to in fd is at the end of the file, otherwise false.

real file_text_read_real(real fd)

On success, the function returns a double-precision numerical value to be read from the current seek position of the file based on the opened file descriptor in fd. Returns zero if the current seek position read was not detected as a numerical value of any kind, or if the function failed for some other reason. The function shall also fail if file descriptor in fd is not opened for reading.

string file_text_read_string(real fd)

On success, the function returns a string value to be read from the current seek position of the file based on the opened file descriptor in fd. The string stops reading and is truncated by the first line feed or carriage return character found. The returned string shall not contain a trailing new line if one was found before reaching the end of the file. The function shall fail if file descriptor in fd is not opened for reading. Returns an empty string on failure.

string file_text_readln(real fd)

On success, the function returns a string value to be read from the current seek position of the file based on the opened file descriptor in fd. The string stops reading and is truncated after the first line feed and/or carriage return character found. The returned string shall contain a trailing new line if one was found before reaching the end of the file. The function shall fail if file descriptor in fd is not opened for reading. Returns an empty string on failure. This function can also be used to change the seek position of the file to the beginning of the next new line after a call to file_text_read_string() or similar functions.

string file_text_read_all(real fd)

If successful, the function reads what is left of the contents of the opened file, (relative to the current seek position), based on the file descriptor referenced in the fd argument. On failure, (or if the current seek position is at the end of the file already), an empty string is returned.

real file_text_open_from_string(string str)

If successful, this function shall create a temporary file and write the string to it found in the str argument. The file Is automatically opened for both reading and writing, and if the function does not fail, it shall return the file descriptor needed to read from and continue to write to it. After writing the string pointed to in str, the seek position is reset to the beginning of the file. On failure, the function returns -1.

The file is not deleted automatically when the file descriptor is closed. To delete the file, before closing the file descriptor, pass the returned file descriptor from this function to a call to file_bin_pathname(), which shall give you the temporary file's pathname. Once that information is copied to a new string, close the file descriptor, and delete the file with a call to file_delete(), should there be the need to do so.

real file_text_close(real fd)

If successful, the function closes an opened file descriptor pointed to by fd, while returning zero on success, and -1 on failure. Always close a file descriptor after you no longer need to read, write, or append data from and/or to it. This function is no different from file_bin_close().