

26. Predefined file routines library

1

The global struct named **files** contains predefined routines for working with files. This clause contains information about using files and the predefined file routines. Like most global objects, the predefined routines in the **files** struct are not meant to be extended with **is also** or **is only**. *See also:* 24.5.

5

26.1 File names and search paths

10

Many of the file routines require a file-name parameter. The following are restrictions on file-name parameters for most routines.

- The file-name shall be the exact path to the file.
- The file-name shall not contain any tildes (~), wild card patterns, or environment variables (including [the path env variable](#)); except for the **files.add_file_type()** routine, which accepts tilde (~), wild cards (*), or [the path env variable](#) as a *file-name* parameter (see 26.3.1).
- Leave the extension off the file-name for files with default extensions, such as .e or .ecom.

268 15

NOTE—**files.add_file_type()** can be used to verify a valid path to a file exists before using any of the file routines.

20

26.2 File handles

269

For every open file, a file handle *struct* exists which contains information about the file. The routine **open()** (see 26.3.4) returns the file handle as a variable of type **file**. The name of the file variable is used in low-level file operations such as the **files.read()**, **files.write()** and **files.flush()** routines (see 26.3).

25

26.3 Low-level file methods

270

This section contains descriptions of the file methods that use file handle *structs*.

30

26.3.1 add_file_type()

Purpose	Get a file name	35
Category	Method	
Syntax	files.add_file_type (<i>file-name</i> : string, <i>file-ext</i> : string, <i>exists</i> : bool): string	40
Parameters	<i>file-name</i> The name of the file to access. A wild card pattern can be used.	
	<i>file-ext</i> The file extension, including the dot (.). This can be empty.	
	<i>exists</i> Whether to check for existence of the file.	45

This assigns a string consisting of *file-name* and *file-ext* to a **string** variable. The *file-name* can contain ~, [the path env variable](#), and * wild cards. The * wild card represents any combination of ASCII characters.

If *file-name* already contains an extension, then *file-ext* is ignored. If *file-ext* is empty, the *file-name* is used with no extension. If *exists* is FALSE, the method returns the file-name string without checking for the existence of the file. Wild cards, ~, and [the path env variable](#) are not evaluated in this case.

50

If *exists* is TRUE, the *e* program checks to see if there is a file that matches the *file-name* in the current directory, based on the following rules.

55

- a) If there is one and only one file that matches the *file-name* pattern, the file's name is returned.
- b) If there is no match in the current directory, then the *path env* directories are searched for the file. If there are multiple matching files in different directories within the *path env* variable, the first one found is returned.
- c) If no matching file can be found or if more than one file is found in a directory that matches a wild card, an error shall be issued.

Syntax example:

```
var fv: string;
fv = files.add_file_type("fname", ".e", FALSE);
```

26.3.2 close()

Purpose	Close a file
Category	Method
Syntax	files.close(<i>file</i>: file-handle)
Parameters	<i>file</i> The file handle of the file to be closed.

This flushes the file buffer and closes the *file*. The file needs to have been previously opened using **open()**.

NOTE—Close a file when no further activity is planned for it to prevent unintentional operations on its contents.

Syntax example:

```
files.close(f_desc);
```

26.3.3 flush()

Purpose	Flush file buffers
Category	Method
Syntax	files.flush(<i>file</i>: file-handle)
Parameters	<i>file</i> The file handle of the file to flush.

This flushes all the operating system buffers associated with *file* to the disk.

File data is buffered in memory and only written to disk at certain times, such as when the file is closed. This method causes data to be written to the disk immediately, which can be useful if two processes are using the same disk file, e.g., to ensure the current data from one process is written to the file before the other process reads from the file.

Syntax example:

```
files.flush(a_file);
```

26.3.4 open()

Purpose	Open a file for reading or writing or both
Category	Method
Syntax	files.open (<i>file-name</i> : string, <i>mode</i> : string, <i>file-role</i> : string): file
Parameters	<i>file-name</i> The name of the file to open. Wild cards, ~, and the path env variable are not allowed in the file name; to use them to select files, see 26.3.1.
	<i>mode</i> The read/write mode for the file. The mode may be one of the following. r — open the file for reading. w — open the file for writing (overwrite the existing contents). rw — open the file for reading and writing (add to the end of the existing contents). a — open the file for appending (add to the end of the existing contents).
	<i>file-role</i> A text description used in error messages about the file.

This opens the file for reading, writing, both reading and writing, or appending, according to mode (*r*, *w*, *rw*, or *a*, respectively) and returns the file [handle](#) of the file. The *file-role* is a description of the file, e.g., "source file".

If the file cannot be opened, an error shall be issued.

Syntax example:

```
var m_file: file;
m_file = files.open("a_file.txt", "r", "Text File");
```

26.3.5 read()

Purpose	Read an ASCII line from a file
Category	Method
Syntax	files.read (<i>file</i> : file- handle , <i>string-var</i> : *string): bool
Parameters	<i>file</i> The file handle of the file that contains the text to read.
	<i>string-var</i> The variable used to hold the read ASCII text.

This reads a line of text from a file into a string variable. The file shall have been opened with **open()**. The line from the file (without the final `\n` newline character) is read into the variable. This method returns `TRUE` on success. If the method cannot read a line (e.g., if the end of the file is reached), it returns `FALSE`. See also: 3.8.1.1 for information about type conversion between scalar types.

NOTE—For performance considerations, use the **for each line in file** action rather than this method.

Syntax example:

```
r_b = files.read(f_desc, m_string);
```

26.3.6 read_lob()

Purpose	Read from a binary file into a list of bits	
Category	Method	
Syntax	files.read_lob (<i>file</i> : file- handle , <i>size-in-bits</i> : int): list of bit	
Parameters	<i>file</i>	The file handle of the file from which to read.
	<i>size-in-bits</i>	The number of bits to read (in multiples of 8).

This reads data from a binary file into a list of bits and returns the list of bits. The file shall already have been opened with **open()**. To read an entire file, use UNDEF as the *size-in-bits*. See also: 3.8.1.1 for information about type conversion between scalar types.

Syntax example:

```
var m_file: file = files.open("a_file.dat", "r", "Data");
var b_l: list of bit;
b_l = files.read_lob(m_file, 32);
```

26.3.7 write()

Purpose	Write a string to file	
Category	Method	
Syntax	files.write (<i>file</i> : file- handle , <i>text</i> : string)	
Parameters	<i>file</i>	The file handle of the file in which to write (w or a).
	<i>text</i>	The text to write to the file.

This adds a string to the end of an existing, open file. A new-line \n is added automatically at the end of the string. The file shall already have been opened using **open()**. If the file is not open, an error message shall be issued.

If the file is opened in write mode (**w**), this method overwrites the existing contents. To avoid overwriting the existing file, open it in append mode (**a**) instead.

NOTE—The >> concatenation operator can be used to append information to the end of a file (see [cross-ref](#)).

Syntax example:

```
files.write(m_file, "Test Procedure");
```

26.3.8 write_lob()

Purpose	Write a list of bits to a binary file	
Category	Method	
Syntax	files.write_lob (<i>file</i> : file- handle , <i>bit-list</i> : list of bit)	
Parameters	<i>file</i>	The file handle of the file in which to write.
	<i>bit-list</i>	A list of bits to write to the file. The size of the list shall be a multiple of 8.

This writes all the bits in the bit list (whose size shall be a multiple of 8) to the end of the file specified by *file*. The file shall already have been opened with **open()**.

Lists of bits are always written in binary format.

Syntax example:

```
var m_file: file = files.open("a_f.dat", "w", "My data");
var b_l: list of bit;
files.write_lob(m_file, b_l);
```

26.3.9 writef()

Purpose	Write to a file in a specified format	
Category	Pseudo-method	
Syntax	files.writef (<i>file</i> : file- handle , <i>format</i> : string, <i>item</i> : exp, ...)	
Parameters	<i>file</i>	The file handle of the file in which to write.
	<i>format</i>	A string containing a standard C formatting mask (see 24.6.3) for each <i>item</i> .
	<i>item</i>	An <i>e</i> expression to write to the file.

This adds a formatted string to the end of the specified file. No newline is automatically added. (Use `\n` in the formatting mask to add a newline.)

The file shall already have been opened with **open()**, otherwise an error shall be issued. If the number of items in the formatting mask is different than the number of item expressions, an error shall be issued.

How the data is written to the file is affected by the **open()** mode (w or a) and whether or not the file already exists, as follows.

- If the file did not previously exist and the w (write) option is used with **open()**, then **writef()** writes the data into a new file.
- If the file did not previously exist and the a (append) option is used with **open()**, then no data is written.
- If the file did previously exist and the w (write) option is used with **open()**, then **writef()** overwrites the contents of the file.

- If the file did previously exist and the a (append) option is used with **open()**, then **writeln()** appends the data to the existing contents of the file.

See also: 24.6.3.

Syntax example:

```
var m_file: file = files.open("a_f.txt", "w", "My data");
var m_i := new m_struct_s;
writeln(m_file, "Type: %s\tData: %s\n", m_i.s_type, m_i.data);
```

26.4 General file routines

This section contains descriptions of the general filing routines. See also: 3.8.1.1 for information about type conversion between scalar types.

26.4.1 file_age()

Purpose	Get a file's modification date
Category	Method
Syntax	files.file_age (<i>file-name</i> : string): int
Parameters	<i>file-name</i> The file whose age is to be found.

This returns the modification date of the file as an integer. This routine can be used to compare the modification dates of files. The integer returned by the routine is not recognizable as a date, but is a unique number derived from the file's modification date. If the modification date includes the time of day, the time is factored into the number the routine returns. Newer files produce larger numbers than older files.

If the file does not exist, an error shall be issued.

Syntax example:

```
var f_data: int;
f_data = files.file_age("f.txt");
```

26.4.2 file_append()

Purpose	Append files
Category	Method
Syntax	files.file_append (<i>from-file-name</i> : string, <i>to-file-name</i> : string)
Parameters	<i>from-file-name</i> The name of the file to append.
	<i>to-file-name</i> The name of the file where the <i>from-file</i> is appended.

This adds the contents of the file named *from-file-name* to the end of the file named *to-file-name*. If either of the files does not exist, an error shall be issued.

Syntax example:

```
files.file_append(f_1, f_2);
```

26.4.3 file_copy()

Purpose	Create a copy of a file
Category	Method
Syntax	files.file_copy (<i>from-file-name</i> : string, <i>to-file-name</i> : string)
Parameters	<i>from-file-name</i> The name of the file to copy.
	<i>to-file-name</i> The name of the (new) copy file.

This makes a copy of the file named *from-file-name*, with the name *to-file-name*. If a file already exists with the *to-file-name*, the contents of that file are replaced by the contents of the file named *from-file-name*. If the file named *from-file-name* does not exist, an error shall be issued.

Syntax example:

```
files.file_copy("file_1.txt", "tmp_file.txt");
```

26.4.4 file_delete()

Purpose	Delete a file
Category	Method
Syntax	files.file_delete (<i>file-name</i> : string)
Parameters	<i>file-name</i> The file to delete.

This deletes the specified file. If the file cannot be found, an error shall be issued.

Syntax example:

```
files.file_delete("run_1.log");
```

26.4.5 file_exists()

Purpose	Check if a file exists
Category	Method
Syntax	<code>files.file_exists(file-name: string): bool</code>
Parameters	<i>file-name</i> The file to check.

This checks if the *file-name* exists in the file system. It returns `TRUE` if the file exists or issues an error if it does not exist. It also returns `TRUE` if the file is a directory. The routine does not check whether the file is readable or not.

NOTE—This routine only checks for the existence of the specified file; for a routine that can check for multiple similarly named files, see 26.3.1.

Syntax example:

```
var f_e: bool;
f_e = files.file_exists("file_1.e");
```

26.4.6 file_extension()

Purpose	Get the extension of a file
Category	Method
Syntax	<code>files.file_extension(file-name: string): string</code>
Parameters	<i>file-name</i> The file name.

This returns a string containing the file extension, which is the sequence of characters after the last period (.) in the file name.

Syntax example:

```
var f_ext: string;
f_ext = files.file_extension("f_1.exe");
```

26.4.7 file_is_dir()

1

Purpose	Check if a file is a directory
Category	Method
Syntax	files.file_is_dir (<i>file-name</i> : string): bool
Parameters	<i>file-name</i> The file to check.

5

10

This returns `TRUE` if the file exists and is a directory. Otherwise, it returns `FALSE` (if the file does not exist or is not a directory).

15

Syntax example:

```
var is_d: bool;
is_d = files.file_is_dir("a_fil");
```

20

26.4.8 file_is_link()

Purpose	Check if a file is a symbolic link
Category	Method
Syntax	files.file_is_link (<i>file-name</i> : string): bool
Parameters	<i>file-name</i> The file to check.

25

This returns `TRUE` if the file exists and is a symbolic link. Otherwise, it returns `FALSE` (if the file does not exist or is not a symbolic link).

35

Syntax example:

```
var is_l: bool;
is_l = files.file_is_link("a_fil");
```

40

26.4.9 file_is_readable()

Purpose	Check if a file is readable
Category	Method
Syntax	files.file_is_readable (<i>file-name</i> : string): bool
Parameters	<i>file-name</i> The file to check.

45

This returns `TRUE` if the file exists and is readable. Otherwise, it returns `FALSE` (if the file does not exist or is not readable).

55

Syntax example:

```
var is_rd: bool;
is_rd = files.file_is_readable("a_fil");
```

26.4.10 file_is_regular()

Purpose	Check if a file is a regular file (not a directory or link)
Category	Method
Syntax	files.file_is_regular (<i>file-name</i> : string): bool
Parameters	<i>file-name</i> The file to check.

This returns `TRUE` if the file exists and is a regular file. Otherwise, it returns `FALSE` (if the file does not exist or is a directory or symbolic link).

Syntax example:

```
var is_rg: bool;
is_rg = files.file_is_regular("a_fil");
```

26.4.11 file_is_temp()

Purpose	Check if a file is a temporary file
Category	Method
Syntax	files.file_is_temp (<i>file-name</i> : string): bool
Parameters	<i>file-name</i> The file to check.

273

This returns `TRUE` if the file is a temporary file per the convention of the host OS. Otherwise, it returns `FALSE`.

Syntax example:

```
var is_tmp: bool;
is_tmp = files.file_is_temp("a_fil");
```

26.4.12 file_is_text()

Purpose	Check if a file is a text file
Category	Method
Syntax	<code>files.file_is_text(file-name: string): bool</code>
Parameters	<i>file-name</i> The file to check.

This returns `TRUE` if the file is a text file (i.e., it contains more than 20% printable characters). Otherwise, it returns `FALSE` (if the file does not exist or it is a not a text file).

Syntax example:

```
var is_txt: bool;
is_txt = files.file_is_text("a_fil");
```

26.4.13 file_rename()

Purpose	Rename a file
Category	Method
Syntax	<code>files.file_rename(from-file-name: string, to-file-name: string)</code>
Parameters	<i>from-file-name</i> The file to rename.
	<i>to-file-name</i> The new file name.

This renames the file named *from-file-name* to *to-file-name*. If any files already exists with *to-file-name*, that file is overwritten by the contents of the file named *from-file-name*.

If the file or directory is not writable, an error shall be issued.

Syntax example:

```
files.file_rename("f_1.exe", "b_1.exe");
```

26.4.14 file_size()

Purpose	Get the size of a file
Category	Method
Syntax	<code>files.file_size(file-name: string): int</code>
Parameters	<i>file-name</i> The file name.

This returns the integer number of bytes in the file. If the file does not exist, an error shall be issued.

Syntax example:

```
var f_s: int;
f_s = files.file_size("a_file.txt");
```

26.4.15 new_temp_file()

Purpose	Create a unique temporary file name
Category	Method
Syntax	files.new_temp_file(): string

This computes a file name (a string with a period (.) at the end). Each file name this routine produces contains the name of the process, so names are unique across processes. The files are saved in the temporary files directory (set per the convention of the host OS).

274

NOTE—This routine only creates the file name; to create a file (with this name), see 26.3.4.

Syntax example:

```
var t_name: string;
t_name = files.new_temp_file()
```

26.4.16 write_string_list()

Purpose	Write a list of strings to a file	
Category	Method	
Syntax	files.write_string_list(file-name: string, strings: list of string)	
Parameters	<i>file-name</i>	The file name in which to write.
	<i>strings</i>	A list of strings to write to the file.

This writes a list of strings into a file. Every string is written on a separate line in the file, with \n appended to the end of the string. If the file already exists, it is overwritten.

If the list of strings contains a NULL, an error shall be issued.

275

NOTE—The >> concatenation operator can be used to append information to the end of a file (see [cross-ref](#)).

Syntax example:

```
var s_list:= {"a string"; "another string"};
files.write_string_list("a_file.txt", s_list);
```

26.5 Reading and writing structs

This section contains descriptions of the file routines that use read structs from files and write structs to files. Structs in e can be read from files and written to files in either binary or ASCII format.

26.5.1 read_ascii_struct()

Purpose	Read ASCII file data into a struct	
Category	Method	
Syntax	<code>files.read_ascii_struct(file-name: string, struct: string): struct</code>	
Parameters	<i>file-name</i>	The name of the (ASCII) file to read.
	<i>struct</i>	The string in which to read the data.

This reads the ASCII contents of *file-name* into a struct of type *struct*, and returns a struct. The struct being read needs to be cast to the correct data type (see 3.8.1). If the file does not exist, an error shall be issued.

Syntax example:

```
var a_str: s_struct;
a_str = files.read_ascii_struct("a_s.out",
    "s_struct").as_a(s_struct);
```

26.5.2 read_binary_struct()

Purpose	Read the contents of a binary file into a struct	
Category	Pseudo-method	
Syntax	<code>files.read_binary_struct(file-name: string, struct: string, check-version: bool): struct</code>	
Parameters	<i>file</i>	The name of the file to read. The file shall have been created by using write_binary_struct() .
	<i>struct</i>	The string in which to read the data.
	<i>check-version</i>	Set to TRUE to compare the contents of the file being read with the definition of the <i>struct</i> in the currently running module. Set to FALSE to allow minor changes.

This reads the binary contents of *file-name* into a *struct* of the specified type, and returns a struct. The struct being read needs to be cast to the correct data type (see 3.8.1).

If *check-version* is FALSE, the routine can run even if the order of fields in the file struct is different from the order of fields in the currently running *e* module. If *check-version* is TRUE, an error shall be issued if the *struct* definition has been changed in any way since the *struct* was written to the file.

Syntax example:

```
var b_str: s_struct;
b_str = files.read_binary_struct("b.out", "s_struct",
    TRUE).as_a(s_struct);
```

26.5.3 write_ascii_struct()

Purpose	Write the contents of a struct to a file in ASCII format
Category	Struct member
Syntax	files.write_ascii_struct (<i>file-name</i> : string, <i>struct</i> : struct, <i>comment</i> : string, <i>indent</i> : bool, <i>depth</i> : int, <i>max-list-items</i> : int)
Parameters	<i>file-name</i> The name of the file in which to write. The default extension is <code>.erd</code> , which stands for <i>e</i> -readable data.
	<i>struct</i> The name of the struct instance to write to the file.
	<i>comment</i> A string for a comment at the beginning of the file.
	<i>indent</i> A Boolean selector for indentation to the struct's field depth.
	<i>depth</i> The number of levels of nested <i>structs</i> to write.
	<i>max-list-items</i> For lists, how many items from each list to write.

This recursively writes the contents of the *struct* to the *file-name* in ASCII format. If the *struct* contains other *structs*, those *structs* are also written to the file. If the number of hierarchical levels contained in the *struct* is greater than the specified *depth*, levels below the *depth* level are represented by ellipses (. . .) in the ASCII file.

The `.erd` default file name extension is automatically added to the file name only if the specified file name has no extension and does not end with a period (.), e.g., `myfile` becomes `myfile.erd`.

This routine does not write any of the *e* program internal *structs*. It can write the **sys** *struct*, but not any pre-defined *structs* within **sys**.

If the file already exists, it is overwritten.

Syntax example:

```
files.write_ascii_struct("a_file.dat", a_str, "my_struct",
    TRUE, 2, 10);
```

26.5.4 write_binary_struct()

Purpose	Write the contents of a struct to a file in binary format
Category	Method
Syntax	files.write_binary_struct (<i>file-name</i> : string, <i>struct</i> : struct)
Parameters	<i>file-name</i> The name of the file in which to write the <i>struct</i> .
	<i>struct</i> The name of the struct instance to write to the file.

This recursively writes the contents of the *struct* to the *file-name* in binary format. If the struct contains other *structs*, those *structs* are also written to the file. If the file already exists, it is overwritten.

Syntax example:

```
files.write_binary_struct("b_file.dat", b_str);
```

1

5

10

15

20

25

30

35

40

45

50

55