
AIMMS Function Reference - File and Directory Functions

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com

File and Directory Functions

AIMMS supports the following functions for accessing disk files and directories:

- DirectoryCopy
- DirectoryCreate
- DirectoryDelete
- DirectoryExists
- DirectoryGetCurrent
- DirectoryGetFiles
- DirectoryGetSubdirectories
- DirectoryMove
- DirectorySelect
- FileAppend
- FileCopy
- FileDelete
- FileEdit
- FileExists
- FileGetSize
- FileMove
- FilePrint
- FileRead
- FileSelect
- FileSelectNew
- FileTime
- FileTouch
- FileView

DirectoryCopy

The procedure `DirectoryCopy` copies one or more directories to a new or other directory.

```
DirectoryCopy(  
    source,      ! (input) scalar string expression  
    destination, ! (input) scalar string expression  
    [confirm]   ! (optional) 0 or 1  
)
```

Arguments:

source

A scalar string expression representing the directories(s) you want to copy. The string may contain wild-card characters such as '*' and '?', allowing you to copy a whole group of directories at once.

destination

A scalar string expression representing the destination directory.

confirm (optional)

An integer value that indicates whether you want to let the user confirm any copy operation that would overwrite existing files. If this argument is omitted, then the default behavior is that files are overwritten without any notice.

Return value:

The procedure returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

Remarks:

If the destination name does not exist, AIMMS will create a directory with the specified name with the same contents as the source directory. If the destination directory does already exist as a directory, AIMMS will copy the contents of the source directory into a directory with the same name as the source directory contained in the destination directory.

See also:

The procedures [DirectoryMove](#), [FileCopy](#), [DirectoryExists](#).

DirectoryCreate

The procedure `DirectoryCreate` creates a new directory on your disk.

```
DirectoryCreate(  
    directoryname    ! (input) scalar string expression  
)
```

Arguments:

directoryname

A scalar string expression representing the new directory name. If the name does not contain a full path, then it is assumed to be relative to the current project directory.

Return value:

The procedure returns 1 if the directory is created successfully. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

Remarks:

If the new directory path contains references to non-existing directories, then the procedure tries to create each of these directories.

See also:

The procedures [DirectoryExists](#), [DirectoryDelete](#).

DirectoryDelete

The procedure `DirectoryDelete` deletes one or more directories from your disk. If any of these directories contain files, then these files are deleted as well.

```
DirectoryDelete(  
    directory,           ! (input) scalar string expression  
    [delete_readonly_files] ! (optional, default 0) scalar expression  
)
```

Arguments:

directory

A scalar string expression representing the directories you want to delete. The string may contain wild-card characters such as '*' and '?', allowing you to delete a whole group of directories at once.

delete_readonly_files

A scalar expression indicating whether read-only files must be deleted without further notice (value \neq 0), or whether the procedure should fail on read-only files (value 0).

Return value:

The procedure returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

See also:

The procedures [DirectoryExists](#), [FileDelete](#).

DirectoryExists

With the procedure `DirectoryExists` you can check whether a specific directory name currently exists.

```
DirectoryExists(  
    directoryname    ! (input) scalar string expression  
)
```

Arguments:

directoryname

A scalar string expression representing a valid directory name. The file name may contain a partial path relative to the project directory, or a full path.

Return value:

The procedure returns 1 if the given directory name exists, or 0 otherwise.

Remarks:

Note that if you want use some static directory name in your model, then you have to specify two slashes behind each directory, as in "c:\\windows\\temp".

See also:

The procedure [DirectoryDelete](#).

DirectoryGetCurrent

The procedure `DirectoryGetCurrent` retrieves the full path of the current project directory.

```
DirectoryGetCurrent(  
    directoryname    ! (output) scalar string parameter  
)
```

Arguments:

directory

A scalar string parameter, that on return will contain the path of the current project directory. The string is always terminated by a directory slash `\`.

Return value:

The procedure returns 1.

See also:

The procedure [DirectorySelect](#).

DirectoryGetFiles

The procedure `DirectoryGetFiles` creates a list of filenames present in a directory.

```
DirectoryGetFiles(
    directory, ! (input) scalar string expression
    filter,    ! (input) scalar string expression
    filenames, ! (output) a one-dimensional string parameter
    recursive ! (optional) default 0
)
```

Arguments:

directory

A scalar string expression representing the directory you want to search. The empty string is interpreted as the current directory.

filter

The pattern file names should match. The empty string is interpreted as all files.

filenames

A one-dimensional string parameter indexed over a subset of the pre-declared set `Integers`. This parameter will be filled with the names of the files matching the pattern as specified in the first argument.

recursive

An optional scalar expression. When zero the procedure `DirectoryGetFiles` doesn't work recursively; it scans only the directory specified, not its subdirectories. When non-zero, these subdirectories will also be searched.

Return value:

The procedure returns the number of files found on success, which may be 0. If it fails, then it returns -1, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

Example:

Using the declarations

```
SET:
    identifier : FileNumbers
    subset of  : Integers
    index      : fn ;
STRING PARAMETER:
    identifier : FileNames
    index domain : (fn) ;
```

the statements

```
DirectoryGetFiles("log", "*.err", Filenames);  
display Filenames ;
```

will result in

```
FileNames := data { 1 : "aimms.err" } ;
```

to be printed in the listing file.

Remarks:

- The directory argument can specify either a relative or an absolute folder path.
- Devices, hidden files, system files, hidden subdirectories and system subdirectories are not searched. On Linux systems, files and subdirectories that start with a '.' are considered hidden files and are not returned in the result.

See also:

- The procedure [DirectoryGetSubdirectories](#) to find the names of the subdirectories in a particular directory.
- The procedures [DirectoryGetCurrent](#) and [DirectorySelect](#) to obtain the current directory and to select a particular directory.

DirectoryGetSubdirectories

The procedure `DirectoryGetSubdirectories` creates a list of subdirectory names present in a directory.

```
DirectoryGetSubdirectories(
    directory,      ! (input) scalar string expression
    filter,        ! (input) scalar string expression
    subdirectorynames, ! (output) a one-dimensional string parameter
    recursive      ! (optional) default 0
)
```

Arguments:

directory

A scalar string expression representing the directory you want to search. The empty string is interpreted as the current directory.

filter

The pattern file names should match. The empty string is interpreted as all files.

subdirectorynames

A one-dimensional string parameter indexed over a subset of the pre-declared set `Integers`. This parameter will be filled with the names of the folders matching the pattern as specified in the first argument.

recursive

An optional scalar expression. When zero the procedure `DirectoryGetSubdirectories` doesn't work recursively; it scans only the directory specified, not its subdirectories. When non-zero, these subdirectories will also be searched.

Return value:

The procedure returns the number of subdirectories found on success, which may be 0. If it fails, then it returns -1, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

Example:

Using the declarations

```
SET:
    identifier : FolderNumbers
    subset of  : Integers
    index     : fn ;
STRING PARAMETER:
    identifier : FolderNames
    index domain : (fn) ;
```

the statements

```
DirectoryGetSubdirectories("", "*.*", FolderNames);  
display FolderNames ;
```

will result in

```
FolderNames := data { 1 : "backup", 2 : "log" } ;
```

to be printed in the listing file.

Remarks:

- The directory argument can specify either a relative or an absolute folder path.
- Hidden and system files and subdirectories are not searched, nor are devices. On Linux systems, files and subdirectories that start with a '.' are considered hidden files and are not searched. The names "." and ".." are never included in the result.

See also:

- The procedure [DirectoryGetFiles](#) to find the names of the files in a particular directory.
- The procedures [DirectoryGetCurrent](#) and [DirectorySelect](#) to obtain the current directory and to select a particular directory.

DirectoryMove

The procedure `DirectoryMove` moves one or more directories to either a new name (a rename) or to another directory.

```
DirectoryMove(  
    source,      ! (input) scalar string expression  
    destination, ! (input) scalar string expression  
    confirm     ! (optional) 0 or 1  
)
```

Arguments:

source

A scalar string expression representing the file(s) you want to move. The string may contain wild-card characters such as “*” and “?”, allowing you to move a whole group of directories at once.

destination

A scalar string expression representing the destination directory.

confirm (optional)

An integer value that indicates whether or not you want to let the user confirm any move operation that would overwrite existing files. If this argument is omitted, then the default behavior is that files are overwritten without any notice.

Return value:

The procedure returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

Remarks:

If the destination name does not exist, AIMMS will move the source directory to the specified position. If the destination directory does already exist as a directory, AIMMS will move the source directory into the (existing) destination directory, retaining the original name of the source directory.

See also:

The procedures [DirectoryCopy](#), [FileMove](#), [DirectoryExists](#).

DirectorySelect

With the procedure `DirectorySelect` you can let the user select an existing directory using Windows' standard directory selection dialog box.

```
DirectorySelect(  
    directoryname, ! (input/output) scalar string parameter  
    [directory,]  ! (optional input) scalar string expression  
    [title]       ! (optional input) scalar string expression  
)
```

Arguments:

directoryname

A scalar string parameter. On return this parameter will represent the selected directory name. If the selected directory is a sub directory below the current project directory, then the directory name will be presented using a relative path. In other cases the directory name is presented using a full path specification. In both cases, the returned directory string is terminated by a \ character.

directory (optional)

A scalar string representing an existing directory. The dialog box will initially select this directory. If omitted, then the current project directory will be used.

title (optional)

A scalar string that is used as the title of the selection dialog box. If this argument is omitted, then a default title is used.

Return value:

The procedure returns 1 if the user did select a directory. If some error occurs or if the user presses the **Cancel** button, then the procedure returns 0.

Remarks:

If `DirectorySelect` returns 0, then the first argument may not contain a valid directory path. So you must always check the return value, and, if it is 0, either abort the current procedure or continue with some default directory name.

See also:

The procedures [FileSelect](#), [DirectoryGetCurrent](#).

FileAppend

The procedure `FileAppend` appends the contents of one file to the end of another file. Both files must be ASCII text files.

```
FileAppend(  
    filename,      ! (input) scalar string expression  
    appendname    ! (input) scalar string expression  
)
```

Arguments:

filename

A scalar string expression representing the file name to which you want to append the contents of the second file.

appendname

A scalar string expression representing the file name that you want to append.

Return value:

The procedure returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

Remarks:

If the first file (the file to which you append) does not exist, then this file will be created. The contents of the appended file will always start on a new line in the resulting file.

See also:

The procedures [FileCopy](#), [FileExists](#).

FileCopy

The procedure `FileCopy` copies one or more files to a new name or to another directory.

```
FileCopy(  
    source,      ! (input) scalar string expression  
    destination, ! (input) scalar string expression  
    [confirm]   ! (optional) 0 or 1  
)
```

Arguments:

source

A scalar string expression representing the file(s) you want to copy. The string may contain wild-card characters such as '*' and '?', allowing you to copy a whole group of files at once.

destination

A scalar string expression representing the destination file name or destination directory.

confirm (optional)

An integer value that indicates whether or not you want to let the user confirm any copy operation that would overwrite existing files. If this argument is omitted, then the default behavior is that files are overwritten without any notice.

Return value:

The procedure returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

See also:

The procedures [FileMove](#), [DirectoryCopy](#), [FileExists](#).

FileDelete

The procedure `FileDelete` deletes one or more files from your disk.

```
FileDelete(  
    filename,           ! (input) scalar string expression  
    [delete_readonly_files] ! (optional, default 0) scalar expression  
)
```

Arguments:

filename

A scalar string expression representing the file(s) you want to delete. The string may contain wild-card characters such as '*' and '?', allowing you to delete a whole group of files at once.

delete_readonly_files

A scalar expression indicating whether read-only files must be deleted without further notice (value \neq 0), or whether the procedure should fail on a read-only file (value 0).

Return value:

The procedure returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

See also:

The procedures [FileExists](#), [DirectoryDelete](#).

FileEdit

The procedure `FileEdit` opens a specific file in the internal AIMMS text file editor. Optionally, you can set the cursor on a specific piece of text within the file.

```
FileEdit(  
    filename,    ! (input) scalar string expression  
    [find]      ! (optional) scalar string expression  
)
```

Arguments:

filename

A scalar string expression representing the file name that you want to edit.

find (optional)

A scalar string expression that is used to position the cursor over a specific piece of text in the file. If this argument is omitted (or if the specified text cannot be found), then the cursor will be positioned at the top of the file.

Return value:

The procedure returns 1 on success, and 0 if it could not open the file in the editor.

Remarks:

If you want to use another external text editor to edit a specific file, then you can use the procedure [Execute](#).

See also:

The procedures [FileView](#), [Execute](#).

FileExists

With the procedure `FileExists` you can check whether a specific file name currently exists.

```
FileExists(  
    filename      ! (input) scalar string expression  
)
```

Arguments:

filename

A scalar string expression representing a valid file name. The file name may contain a partial path relative to the project directory, or a full path.

Return value:

The procedure returns 1 if the given file name exists, and 0 otherwise.

Remarks:

Note that if you want use some static file name in your model, then you have to specify two slashes behind each directory, as in "c:\\windows\\temp\\filename.dat"

See also:

The procedure `FileDelete`

FileGetSize

The procedure `FileGetSize` retrieves the size on disk of an existing file.

```
FileGetSize(  
    filename,    ! (input) scalar string expression  
    fileSize    ! (output) scalar numerical identifier  
)
```

Arguments:

filename

A scalar string expression representing an existing file name.

fileSize

A scalar identifier to hold the size of the file, or -1 if the size could not be retrieved.

Return value:

The procedure returns 1 on success. If it failed to retrieve the file size, then it returns 0 and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

See also:

The procedure `FileExists`.

FileMove

The procedure `FileMove` moves one or more files to either a new name (a re-name) or to another directory.

```
FileMove(  
    source,      ! (input) scalar string expression  
    destination, ! (input) scalar string expression  
    [confirm]   ! (optional) 0 or 1  
)
```

Arguments:

source

A scalar string expression representing the file(s) you want to move. The string may contain wild-card characters such as “*” and “?”, allowing you to move a whole group of files at once.

destination

A scalar string expression representing the destination file name or destination directory.

confirm (optional)

An integer value that indicates whether or not you want to let the user confirm any move operation that would overwrite existing files. If this argument is omitted, then the default behavior is that files are overwritten without any notice.

Return value:

The procedure returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

See also:

The procedures [FileCopy](#), [DirectoryMove](#), [FileExists](#).

FilePrint

The procedure `FilePrint` prints a specific text file using the currently selected printer.

```
FilePrint(  
    filename      ! (input) scalar string expression  
)
```

Arguments:

filename
A scalar string expression representing the text file that you want to print.

Return value:

The procedure returns 1 on success, and 0 if it could not print the file.

Remarks:

The file is printed using the paper and font settings that are specified in the **Text Printing** dialog box, which is accessible from the Settings menu.

See also:

The procedure [FileEdit](#).

FileRead

With the procedure `FileRead` you can read the contents of a file into a string parameter.

```
FileRead(  
    filename      ! (input) scalar string expression  
)
```

Arguments:

filename

A scalar string expression representing a valid file name. The file name may contain a partial path relative to the project directory, or a full path.

Return value:

The procedure returns a string containing the contents of the file.

Remarks:

- This procedure will not automatically reread a file when its contents has changed. It is therefore better to use it in a procedure than in a parameter definition.
- In case the file does not exist, no error message will be returned and the result will be the empty string. In case there is any doubt the file exists it is advised to first check using the procedure `FileExists`.

See also:

The procedure `FileExists`.

FileSelect

With the procedure `FileSelect` you can let the user select an existing file name using Windows' standard file selection dialog box. Usually you use this procedure to select some input file (i.e. a file for reading), because other than `FileSelectNew`, this procedure only allows the user to select existing files.

```
FileSelect(  
    filename,      ! (input/output) scalar string identifier  
    [directory,]  ! (optional) scalar string expression  
    [extension,]  ! (optional) scalar string expression  
    [title]       ! (optional) scalar string expression  
)
```

Arguments:

filename

A scalar string identifier holding the file name that the user selected. If on entry this string represents a valid file name, then this file name is used to initialize the dialog box.

directory (optional)

A scalar string representing an existing directory. The dialog box will initially only show the files that are located in this directory. If this argument is omitted, then the current project directory will be used.

extension (optional)

A scalar string representing a file extension. The dialog box will initially only show those files that match this extension. If this argument is omitted, then all files are shown.

title (optional)

A scalar string that is used as the title of the selection dialog box. If this argument is omitted, then a default title is used.

Return value:

The procedure returns 1 if the user actually has selected a file. If some error occurs or if the user presses the **Cancel** button, the procedure returns 0.

Remarks:

If `FileSelect` returns 0, then the first argument may not contain a valid file name. So you must always check the return value, and, if it is 0, either abort the current procedure or continue with some default file name.

See also:

The procedure `FileSelectNew`.

FileSelectNew

With the procedure `FileSelectNew` the user can select a new (or existing) file using Windows' file selection dialog box. Usually it is used to select an output file (i.e. for writing), because other than `FileSelect`, this procedure allows you to specify new file names. If an existing file name is selected, a warning will be displayed. The procedure does not create any files on disk or make any changes to existing files. It only returns the file name selected by the user.

```
FileSelectNew(  
    filename,      ! (input/output) scalar string identifier  
    [directory,]  ! (optional) scalar string expression  
    [extension,] ! (optional) scalar string expression  
    [title]       ! (optional) scalar string expression  
)
```

Arguments:

filename

A scalar string identifier holding the file name that the user specified. If on entry this string represents a valid file name, then this file name is used to initialize the dialog box.

directory (optional)

A scalar string representing an existing directory. The dialog box will initially only show the files that are located in this directory. If this argument is omitted, then the current project directory will be used.

extension (optional)

A scalar string representing a file extension. The dialog box will initially only show those files that match this extension. If this argument is omitted, then all files are shown.

title (optional)

A scalar string that is used as the title of the selection dialog box. If this argument is omitted, then a default title is used.

Return value:

The procedure returns 1 if the user actually has selected a file. If some error occurs or if the user presses the **Cancel** button, the procedure returns 0.

Remarks:

If `FileSelectNew` returns 0, then the first argument may not contain a valid file name. So you must always check the return value, and, if it is 0, either abort the current procedure or continue with some default file name.

See also:

The procedure `FileSelect`.

FileTime

The procedure `FileTime` retrieves the last modification time of an existing file.

```
FileTime(  
    filename,    ! (input) scalar string expression  
    file_time    ! (output) scalar string identifier  
)
```

Arguments:

filename

A scalar string expression representing an existing file name.

file_time

A scalar string identifier to hold the file modification time of the specified file. This time is represented using AIMMS' standard date and time format: "YYYY-MM-DD hh:mm:ss"

Return value:

The procedure returns 1 on success. If it failed to retrieve the file time, then it returns 0 and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

See also:

The procedure `FileExists`.

FileTouch

The procedure FileTouch changes the modification time of a file.

```
FileTouch(  
    filename,    ! (input) scalar string expression  
    [newtime]    ! (optional) scalar string expression  
)
```

Arguments:

filename

A scalar string expression representing an existing file name.

newtime

This time is represented using AIMMS' standard date and time format: "YYYY-MM-DD hh:mm:ss". If omitted the modification time of the file is set to the current time.

Return value:

The procedure returns 1 on success. If it failed to set the file time, then it returns 0 and the pre-defined identifier CurrentErrorMessage will contain a proper error message.

FileView

The procedure `FileView` opens a specific file in the internal AIMMS text file viewer. Optionally, you can highlight a specific piece of text within the file.

```
FileView(  
    filename,    ! (input) scalar string expression  
    find        ! (optional) scalar string expression  
)
```

Arguments:

filename

A scalar string expression representing the file name that you want to edit.

find (optional)

A scalar string expression that is used to position the cursor over a specific piece of text in the file. If this argument is omitted (or if the specified text cannot be found), then the cursor will be positioned at the top of the file.

Return value:

The procedure returns 1 on success, and 0 if it could not open the file in the viewer.

Remarks:

If you want to use another external text editor to view a specific file, then you can use the procedure [Execute](#).

See also:

The procedures [FileEdit](#), [Execute](#).