

## AbsFile Version 1.0.12

# How to use this Xtra

To be able to use AbsFile, copy the file AbsFile.x32 into the file "xtras" folder, which is in the Director installation folder.

To obtain a summary of the xtra's functionality, type the following into Director's message window:  
Put xtra("AbsFile").interface()

To create an instance of AbsFile, use the following code:  
Dot address (Director 7 or higher)  
AbsObj=xtra("AbsFile").new()

After having created an instance, you will be able to use all of the xtra's functions. With the exception of AbsFileRegister(), all of the functions require a running instance of the xtra prior to being used, in accordance with the following example:

Dot address (Director 7 or higher)  
AbsObj.openFile("\*.txt")

When you do no longer need the xtra, you should release its allocated memory in the following way:  
Dot address (Director 7 or higher)  
AbsObj=VOID

## AbsFileRegister(string key)

Registers the xtra for the current session. At the time of the purchase of a full version of the xtra, you will be provided with an unlock key. This key should be passed as a parameter to this function. Once the xtra has been registered, you will have access to all the functions, without limitation.

### Parameter:

**Key:** a string. A string of characters provided to you when you purchase a licence for the AbsFile xtra.

### Returns:

Nothing

### Note:

This function does not require an active instance of the AbsFile xtra.

You only need to register the xtra once for each Director session. The registration function must be called prior to first use of any of the other functions of the xtra.

If you do not include a registration key, the xtra is still entirely functional. However, a dialog box will be displayed each time an instance of the xtra is created.

### Example:

```
AbsFileRegister("String")
```

---

## Creation of an instance and file opening

---

### **new()**

Creation of an instance of the xtra.

All the functions of the xtra (except `AbsFileRegister`) require an active instance of the xtra in order to be executed.

#### **Returns:**

An instance of the xtra

#### **Example:**

```
gAbsFile=xtra("AbsFile").new()
```

### **Open(string pathName, int openMode, int creationMode)**

Opens a file.

#### **Parameters:**

**PathName:** A string. The path of the file to open. The absolute path must be provided.

**OpenMode:** an integer which specifies file opening mode

- 0 read
- 1 read/write
- 2 shareRead
- 4 shareWrite

Combinations are possible. E.g.: 3 (2+1) opens a file in read mode and permits read sharing.

**CreationMode:** an integer. Specifies the way the xtra is to react regarding the presence or absence of the file specified in the `pathName` parameter

- 0 An error occurs if the file does not exist.
- 1 If necessary, the file is created.
- 2 The file, if it exists, is reset to zero

Combinations are possible. The values 1 and 2 are only valid if the `OpenMode` parameter is 1 or 4 (or all combinations incorporating one of these two values)

#### **Returns:**

The symbol `#ok` if the file is opened correctly.

The symbol `#err` if an error occurs. In this case, consult `GetError()` to obtain the error code.

#### **Example:**

```
AbsObj=xtra("AbsFile").new()
```

```
Put AbsObj.open("c:\file.dat",0,0)
-- #ok
AbsObj=VOID
```

## Close()

Closes the open file. This function is called automatically, if necessary, when an instance is destroyed.

Once a file has been opened by a call to the Open() function, it is impossible to open another file within the same instance of the xtra. If you need to open a second file, you can either create another instance of AbsFile or close the previously opened file, by calling the Close() function.

### Returns:

The symbol #ok if the file is closed correctly.

The symbol #err if an error occurs. In this case, consult GetError() to obtain the error code.

### Example:

```
AbsObj=xtra("AbsFile").new()
-- open an existing file in read mode
if AbsObj.Open("c:\file.dat", 0,0)=#err then
    put AbsObj.GetError()
end if
--read or write some data into the file
AbsObj.close()
--once the first file is closed, open another file with the same instance
if AbsObj.Open("c:\anotherFile.def", 0,0)=#err then
    put AbsObj.GetError()
end if
```

---

## OPEN/SAVE DIALOG

---

### OpenDialog(string extension)

Displays the Windows file open dialog and opens the file in read mode.

#### Parameter:

**Extension:** a string. Specifies the extension of the files to be displayed. If the string is empty, all files are displayed.

#### Returns:

A string if the user has selected a file and has clicked on "OK". The string specifies the path and the selected filename. The file is immediately opened in read mode. It is not necessary to call the Open() function. The symbol #cancel if the user has clicked on "Cancel". The symbol #err if an error has occurred. Consult GetError() to obtain the error code.

**Note:**

If the user enters a filename without specifying an extension, the file extension parameter is automatically added.

**Example:**

```
AbsObj=extra("AbsFile").new()  
Put AbsObj.openDialog("*.dat")  
-- "c:\file.dat"  
AbsObj=VOID
```

## SaveDialog(string extension)

Displays the Windows save file dialog box and opens the file in read/write mode.

**Parameter:**

**Extension:** a string. Specifies the extension of the files to be displayed. If the string is empty, all files are displayed.

**Returns:**

A string. If the user has selected a file and has clicked on "OK", the string specifies the path and the selected filename. The file is immediately opened in read/write mode. It is not necessary to call the Open() function. The symbol #cancel if the user has clicked on "Cancel". The symbol #err if an error has occurred. Consult GetError() to obtain the error code.

**Note:**

If the user enters a filename without specifying an extension, the file extension parameter is automatically added.

**Example:**

```
AbsObj=extra("AbsFile").new()  
Put AbsObj.saveDialog("*.dat")  
-- "c:\file.dat"  
AbsObj=VOID
```

## FileOpenDialog(string title, string filter, string defaultExt, string initDir, string defaultName, integer multiSelect, integer openFile)

Displays an open file dialog.  
The file(s) selected by the user must exist.

## Parameters:

**title:** a string. Specifies the title of the dialog box. If the string is empty, the Windows default title is displayed.

**filter:** a string. Specifies the type of file to be displayed.

This parameter can be used to specify 1 or several file types.

If the string is empty, all files will be displayed.

If the string is not empty, it must be composed of Type/Extension pairs, separated by the "|" character. "Text file|.txt" will only display files with the extension "txt". The user will see the string "Text file" in the dialog box.

"Text file|.txt|Word file|.doc" will display either files with the extension "txt" or with the extension "doc". The user has to choose the type of files to display using the drop down list,.

It is also possible to display several types of file simultaneously. To achieve this, specify several extensions together, separated by the ";" character. "Text file|.txt;\*.doc" will display files with "txt" or "doc" extensions.

You can combine these various possibilities:

"Text file|.txt;\*.doc|Music file|.wav|Video file|.avi;\*.mov" will display:

Files with extension "txt" or "doc"

Files with extension "wav"

Files with extension "avi" or "mov"

**defaultExt:** a string. The extension that is added to the filename automatically if the user has not specified one. If it is an empty string, no extension is added.

**InitDir** Specifies the dialog box's initial folder. If the string is empty, Windows selects the last folder browsed by the user.

**DefaultName** Default filename proposed in the file text box. The string can be empty.

**multiSelect** If not zero, the user is allowed to select several files.

**openFile** If not zero, the file is automatically opened if the user chooses "OK". The value is ignored if the "multiselect" parameter is used

## Returns:

A string. If the user selects a file and clicks on "OK", the string returns the full path and the given filename.

A list. If the multiselect parameter is not zero and the user has selected several files, the full path and filename of each file is returned in a list.

The symbol #cancel if the user has clicked on "Cancel".

The symbol #err if an error has occurred. Consult GetLastError() to obtain the error code.

## Note:

1) If the "openFile" parameter does not equal 0 and the "multiSelect" parameter equals 0, the file is opened automatically in read mode.

2) The user can select several files by using the <Shift> and/or <control> keys.

### Example:

1) Prompt to select any type of file. The path and the filename are returned in a string.

```
AbsObj=extra("AbsFile").new()  
Put AbsObj.fileOpenDialog("Select a file", "", "", "", "", 0,0)  
-- "c:\file.dat"  
AbsObj=VOID
```

2) Prompt to select a "wav" type file. If the user enters a filename without an extension, ".wav" will be added automatically to the end of the name. The path and the filename are returned in a string. This file is opened automatically in read mode if the user clicks on "OK".

```
AbsObj=extra("AbsFile").new()  
Put AbsObj.fileOpenDialog("Select a file", "Wav file|*.wav", "wav", "", "", 0,1)  
-- "c:\file.wav"  
AbsObj=VOID
```

3) Prompt to select a text file or a music file. The user can select a "txt" or "doc" file if the file type chosen is "Text file" or a "wav" file if "Music file" is chosen. The path and the filename are returned in a string.

```
AbsObj=extra("AbsFile").new()  
Put AbsObj.fileOpenDialog("Select a file", "Text file|*.txt ;*.doc|Music  
file|*.wav", "", "", "", 0,1)  
-- "c:\file.doc"  
AbsObj=VOID
```

4) Prompt to select one or several "txt" type files. If the user selects one file, the path and the filename are returned in a string. If the user selects several files simultaneously, the path and filename of each file is returned in a list.

```
AbsObj=extra("AbsFile").new()  
Put AbsObj.fileOpenDialog("Select one or several files", "Text file|*.txt", "",  
"", "", 1,0)  
-- ["c:\aFile.txt", "c:\anotherFile.txt"]  
AbsObj=VOID
```

## **FileSaveDialog(string title, string filter, string defaultExt, string initDir, string defaultName, integer multiSelect, integer openFile)**

Displays a save file dialog.

### Parameters:

**title:** a string. Specifies the title of the dialog box. If the string is empty, the Windows default title is displayed.

**filter:** a string. Specifies the type of file to be displayed.

This parameter can be used to specify 1 or several file types. If the string is empty, all files will be displayed.

If the string is not empty, it must be composed of Type/Extension pairs, separated by the "|" character. "Text file|.txt" will only display files with the extension "txt". The user will see the string "Text files" in the dialog box. "Text file|.txt|Word file|.doc" will display either files with the extension "txt" or with the extension "doc". The user has to choose the type of files to display using the drop down list.

It is also possible to display several types of file simultaneously. To achieve this, specify several extensions together, separated by the ";" character. "Text file|.txt ;\*.doc" will display files with the "txt" and "doc" extensions.

You can combine these various possibilities:

"Text file|.txt ;\*.doc|Music file|.wav|Video file|.avi ;\*.mov" will display:

Files with extension "txt" or "doc"

Files with extension "wav"

Files with extension "avi" or "mov"

**defaultExt:** a string. The extension that is added to the filename automatically if the user has not specified one. If defaultExt is empty, no extension is added.

**InitDir** Specifies the dialog box's initial folder. If the string is empty, Windows selects the last folder browsed by the user.

**DefaultName** Default filename proposed in the file text box. The string can be empty.

**multiSelect** If not zero, the user is allowed to choose several files.

**openFile** If not zero, the file is automatically opened if the user chooses "OK". The value is ignored if the "multiselect" parameter is used.

### Returns:

A string. If the user selects a file and clicks on "OK", the string returns the full path and the given filename.

A list. If the multiselect parameter is not zero and the user has selected several files, the full path and filename of each file is returned in a list.

The symbol #cancel if the user has clicked on "Cancel".

The symbol #err if an error has occurred. Consult GetLastError() to obtain the error code.

### Note:

1) If the "openFile" parameter does not equal 0 and the "multiSelect" parameter equals 0, the file is opened automatically in read/write mode.

2) The user can select several files by using the <Shift> and/or <control> keys.

3) If the file specified by the user exists already, a warning message will be displayed:

"The file already exists. Would you like to replace it?". If the user clicks "OK", the existing file is not actually deleted.

### Example:

1) Prompt to choose any type of file. The path and the filename are returned in a string.

```
AbsObj=extra("AbsFile").new()
Put AbsObj.fileSaveDialog("Choose a file", "", "", "", "", 0,0)
-- "c:\file.dat"
AbsObj=VOID
```

2) Prompt to choose a "wav" type file. If the user enters a filename without an extension, ".wav" will be added automatically to the end of the name. The path and the filename are returned in a string. This file is opened automatically in read/write mode if the user clicks on "OK".

```
AbsObj=extra("AbsFile").new()
Put AbsObj.fileSaveDialog("Choose a file", "Wav file|*.wav", "wav",
"", "", 0,1)
-- "c:\file.wav"
AbsObj=VOID
```

3) Prompt to choose a text file or a music file. The user can select a "txt" or "doc" file if the file type chosen is "Text file" or a "wav" file if "Music file" is chosen. The path and the filename are returned in a string.

```
AbsObj=extra("AbsFile").new()
Put AbsObj.fileSaveDialog("Choose a file", "Text file|*.txt ;*.doc|Music
file|*.wav", "", "", "", 0,1)
-- "c:\file.doc"
AbsObj=VOID
```

4) Prompt to choose one or several "txt" type files. If the user chooses one file, the path and the filename are returned in a string. If the user chooses several files simultaneously, the path and filename of each file is returned in a list.

```
AbsObj=extra("AbsFile").new()
Put AbsObj.fileSaveDialog("Choose one or several files", "Text file|*.txt", "",
"", "", 1,0)
-- ["c:\aFile.txt", "c:\anotherFile.txt"]
AbsObj=VOID
```

## FolderDialog(string title)

Displays a dialog for selecting a Windows folder.

### Parameter:

**Title:** a string. The title of the dialog box.

### Returns:

A string. The path and the name of the selected folder.  
The symbol #cancel if the user has clicked on "Cancel".  
The symbol #err if an error has occurred. Consult GetLastError() to obtain the error code.

**Note:**

- 1) Only valid folders can be selected. "My Computer", "Network Neighbourhood", etc. cannot be selected.
- 2) The path always ends with "\"

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.folderDialog("Select a folder")
-- "c:\program files\"
AbsObj=VOID
```

## ComputerDialog(string title)

Displays a dialog for selecting a computer.

**Parameter:**

**Title:** A string. The title of the dialog box.

**Returns:**

A string. The name of the selected computer.  
The symbol #cancel if the user has clicked on "Cancel".  
The symbol #err if an error has occurred. Consult GetLastError() to obtain the error code.

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.computerDialog("Select a computer")
-- "\\computer"
AbsObj=VOID
```

## PrinterDialog(string title)

Displays a dialog for selecting a printer. The user can select a printer on the local computer or over the network.

**Parameter:**

**Title:** a string. The name of the dialog box.

**Returns:**

A string. The path and name of the selected printer.  
The symbol #cancel if the user has clicked on "Cancel".  
The symbol #err if an error has occurred. Consult GetLastError() to obtain the error code.

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.computerDialog("Select a computer")
-- "\\computer\sharedprinter"
AbsObj=VOID
```

---

## Configuration Functions

---

### **SetIntegerFormat(integer zeroForWindows)**

Sets the format for the reading and writing of integers.

Use SetIntegerFormat(0) (default format) for the little endian format (PC Windows).

Use SetIntegerFormat(1) (default format) for the big endian format (Other systems).

The function applies to short (16 bit) and long (32 bit) integers, signed and unsigned.

Calling this function is necessary in order to read/write certain non PC based files. E.g: MP3 files, TIFF files, etc.

#### **Parameter:**

ZeroForWindows:

0 – read/write in little endian format

1 – read/write in big endian format

#### **Returns:**

nothing

#### **Note:**

1) By default, all read and write functions use the little endian format.

2) It is up to you to determine the best format to use. Nothing can indicate the format that has been used.

#### **Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.setIntegerFormat(0)
-- "\\computer\sharedprinter"
AbsObj=VOID
```

---

## Read Functions

---

Values are read from the current position of the file pointer. The file is read sequentially. These functions return #err if an error occurs or #eof if the read head goes beyond the end of the file.

## ReadByte()

Reads a signed-byte and advances the pointer by one byte.

### Returns:

A signed integer. (-128 → 127)

The symbol #eof if the file pointer is beyond the end of the file.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readByte()
-- 50
AbsObj=VOID
```

## ReadUByte()

Reads a byte and advances the pointer by one byte.

### Returns:

A unsigned integer. (0 → 255)

The symbol #eof if the file pointer is beyond the end of the file.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readUByte()
-- -50
AbsObj=VOID
```

## ReadShort()

Reads a 16 bit integer and advances the pointer by 2 bytes.

### Returns:

A 16 bit signed integer. (-32768 → 32767)

The symbol #eof if the file pointer is beyond the end of the file.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
```

```
Put AbsObj.readShort()
-- 4830
AbsObj=VOID
```

## ReadUShort()

Reads a 16 bit integer and advances the pointer by 2 bytes.

### Returns:

A 16 bit unsigned integer. (0 → 65535)  
The symbol #eof if the file pointer is beyond the end of the file.  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readUShort()
-- -7520
AbsObj=VOID
```

## ReadInt()

Reads a 32 bit integer and advances the pointer by 4 bytes.

### Returns:

A 32 bit signed integer. (-2147483648 → 2147483647)  
The symbol #eof if the file pointer is beyond the end of the file.  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readInt()
-- 16485440
AbsObj=VOID
```

## ReadUInt()

Reads a 32 bit integer and advances the pointer by 4 bytes.

### Returns:

An unsigned value (0 → 4294967295)  
From 0 to 2147483647 returns an integer  
From 2147483648 to 4294967295 returns a float value.  
This is because integers in Director cannot be greater than 2147483647  
( $2^{31} - 1$ )

The symbol #eof if the file pointer is beyond the end of the file.  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readUInt()
-- -84659660
AbsObj=VOID
```

## ReadFloat()

Reads a 64 bit IEEE signed decimal (Director internal format for decimals) and advances the pointer by 8 bytes.

**Returns:**

A float value (+/- 2.2250738585072014 E-308 → +/- 1.7976931348623158 E+308)

The symbol #eof if the file pointer is beyond the end of the file.  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readFloat()
-- 1.00059636159647e33
AbsObj=VOID
```

## ReadRect()

Reads a rectangle in Director format and advances the pointer by 16 bytes.

**Returns:**

A Director rect (rect(top,left,bottom,right))  
The symbol #eof if the file pointer is beyond the end of the file.  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readRect()
-- rect(10,50,100,200)
AbsObj=VOID
```

## ReadPoint()

Reads a point in Director format and advances the file pointer by 8 bytes.

### Returns:

A Director point (point(y,x))

The symbol #eof if the file pointer is beyond the end of the file.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readPoint()
-- point(50,100)
AbsObj=VOID
```

## ReadSymbol()

Reads a symbol (see WriteSymbol function). The file pointer advances by a number of bytes equal to 2 + the text length of the symbol (#symbol advances by 8 bytes = 2 + 6)

### Returns:

A symbol.

The symbol #eof if the file pointer is beyond the end of the file.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readSymbol()
-- #text
AbsObj=VOID
```

## ReadBinary(integer nBytes)

Reads a sequence of bytes and places them in a string of characters. The pointer is moved by nBytes.

### Parameter:

**nBytes:** an integer. The number of characters to read.

If nBytes<0, the file is read to its end.

### Returns:

A string. A zero is added to the end of the string.

The symbol #eof if the file pointer is beyond the end of the file.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Note:**

The readBinary function interprets the byte sequence read as a string of characters.

For Director a string of characters always ends with a zero. If a 0 is found within the bytes read, Director will consider that the string stops at the point where the 0 was found. However, the string in memory still will be the one that was initially read, i.e:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.writestring("Hello",#nil)
-- 5
put AbsObj.writeByte(0)
-- 1
put AbsObj.write("world",#nil)
-- 5
AbsObj.setPosition(0,0)
Str=AbsObj.readBinary(11)
Put Str
-- "Hello"
Put Str.length
-- 11
Put Str.char[10]
-- "l"
```

The display of the full string stops at 5 characters, because the 6th is a zero. However, the length remains 11, and the 10th character is the "l" of "world".

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readBinary(5)
-- "Hello world"
AbsObj=VOID
```

## ReadString(integer maxChar)

Reads a string of characters, with maxChar being the maximum number of characters to read. Advances the file pointer by the number of bytes actually read.

**Parameter:**

MaxChar: an integer. The maximum number of characters to read.

If `maxChar < 0`, `readString` reads the file from its current position to the end of the file or until the next 0, CR (carriage return), (line Feed), CR + LF and Ctrl+Z (end of text).

Reading stops at the 1st 0, CR, LF or Ctrl+Z, even if `maxChar` is `< 0` or not.

**Returns:**

A string. The string read. A zero is always added to the end of the string.

The symbol `#eof` if the file pointer is beyond the end of the file.

The symbol `#err` if an error occurs. Consult `GetError()` to obtain the error code.

**Note:**

Whatever the number of characters requested (`maxChar`), reading stops at the 1st zero, CR, LF (line feed) or Ctrl+Z, or if the end of the file has been reached.

**Example:**

```
AbsObj=xtra("AbsFile").new()  
Put AbsObj.open("c:\file.dat",0,0)  
-- #Ok  
Put AbsObj.readString(11)  
-- "Hello World"  
AbsObj=VOID
```

## ReadText(integer maxChar)

Reads a string, with `maxChar` being the maximum number of characters to read.

Reads a string of characters. Reading stops at the 1st zero, Ctrl+Z (end of text) or end of file. Unlike `readString`, this function reads the CR and LF characters.

Advances the file pointer by the number of bytes actually read.

**Parameter:**

`MaxChar`: the maximum number of characters to read.

If `maxChar < 0`, reads file from its current position to the end or until the next 0.

**Returns:**

A string: the string read. A zero is always added to the end of the string.

The symbol `#eof` if the file pointer is beyond the end of the file.

The symbol `#err` if an error occurs. Consult `GetError()` to obtain the error code.

**Note:**

Whatever the number of characters requested (`maxChar`), reading stops at the 1st zero or if the end of the file has been reached.

Unlike readString, this function reads the CR and LF characters.

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readText(11)
-- "Hello World"
AbsObj=VOID
```

## ReadList(string listName)

Reads a list, saved using WriteList, as listName.

The file does not have to be positioned at the start of the list. The function automatically finds the list by its name.

The function is able to read nested lists as well as lists of properties.

**Parameter:**

**ListName:** the name of the list previously written using writeList(). Attention: the name is case sensitive. "aName" is different from "aname".

**Returns:**

A list. The previously written list.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

## BytesReaded()

Indicates the number of bytes read by the last read function called.

**Returns:**

An integer: the number of bytes read if less than or equal to 2147483647

A float value: the number of bytes read if greater than 2147483647

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readInt()
-- 59
Put AbsObj.bytesReaded()
-- 4
AbsObj=VOID
```

---

## Write Functions

---

## WriteByte(integer value)

Writes a signed byte and advances the pointer by 1 byte. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

### Parameter:

**Value:** an integer from -128 to +127. If value cannot be contained in a byte (less than -128 or greater than 127, only the little-endian is written)

### Returns:

An integer if the function executed successfully. The integer indicates the number of bytes written.  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()  
Put AbsObj.open("c:\file.dat",1,3)  
-- #Ok  
Put AbsObj.writeByte(-10)  
-- 1  
AbsObj=VOID
```

## WriteUByte(integer value)

Writes an unsigned byte and advances the pointer by 1 byte. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

### Parameter:

**Value:** an integer from 0 to 255

### Returns:

An integer if the function executed successfully. The integer indicates the number of bytes written.  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()  
Put AbsObj.open("c:\file.dat",1,3)  
-- #Ok  
Put AbsObj.writeUByte(10)  
-- 1  
AbsObj=VOID
```

## WriteShort(integer value)

Writes a 16 bit signed integer and advances the pointer by 2 bytes. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

### Parameter:

**Value:** an integer from -32768 to +32767

### Returns:

An integer if the function executed successfully. The integer indicates the number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.writeShort(-4580)
-- 2
AbsObj=VOID
```

## WriteUShort (integer value)

Writes a 16 bit unsigned integer and advances the pointer by 2 bytes. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

### Parameter:

**Value:** an integer from 0 to 65535

### Returns:

An integer if the function executed successfully. The integer indicates the number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.writeUShort(4620)
-- 2
AbsObj=VOID
```

## WriteInt(integer value)

Writes a 32 bit signed integer and advances the pointer by 4 bytes. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

### Parameter:

**Value:** an integer from -2147483648 to 2147483647

### Returns:

An integer if the function executed successfully. The integer indicates the number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.writeInt(-16777650)
-- 4
AbsObj=VOID
```

## WriteUInt(float value)

Writes a 32 bit unsigned integer and advances the pointer by 4 bytes. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

### Parameter:

**Value:** an integer from 0 to 4294967295

### Returns:

An integer if the function executed successfully. The integer indicates the number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Example:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.writeUInt(2789684520)
-- 4
AbsObj=VOID
```

## WriteFloat(float value)

Writes an IEEE 64 bit signed decimal (internal Director decimal format) and advances the pointer by 8 bytes. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

**Parameter:**

**Value:** a floating-point number

**Retourne**

An integer: the number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.writeFloat(1536.265542)
-- 8
AbsObj=VOID
```

## WriteSymbol(symbol value)

Writes a symbol. In the file, the symbol is stored as a string preceded by # and followed by 0. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

The file pointer is advanced by a number of bytes equal to 2 + the text length of the symbol

**Parameter:**

**Value:** a Director symbol

**Returns:**

An integer if the function executed successfully. The integer indicates the number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.writeSymbol(#symbol)
-- 8
AbsObj=VOID
```

## Write(any)

Writes one or several Director values. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

The file pointer is advanced by a varying number of bytes depending on the variable(s) being written.

**Parameter:**

**Any:** 1 or several Director parameters, separated by commas.

Suitable types are:

Integer, float, string, symbol, point, rect, list and proplist.

List elements must also be of suitable types (Integer, float, string, symbol, point, rect, list and proplist).

**Returns:**

An integer if the function executed successfully. The integer indicates the number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Note:**

Strings are written with a final zero.

**Example:**

1) Simple usage of the write() function:

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.write(55)
-- 4
Put AbsObj.write("Hello")
-- 6
AbsObj=VOID
```

2) Usage with several parameters

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.write(55,"Hello",point(10,20))
-- 18
AbsObj=VOID
```

## **WriteString(string aString, symbol terminator)**

Writes a string and allows definition of the termination character. If the file is too small to contain the value, or if the file pointer is at the end of the file, the file is increased in size.

**Parameters:**

**aString:** the string of characters to write.

**terminator:** a symbol.

#zero ou #o	a zero is stored at the end of the string (identical to Write(string))
#crlf	a carriage return + line feed (format PC) is added
#cr	a carriage return character is added
#end ou #z	an end of text marker is added
#- ou #nil	nothing is added

**Returns:**

An integer if the function executed successfully. The integer indicates the number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.writeString("Hello World",#zero)
-- 12
AbsObj=VOID
```

## WriteList(string listName, list theList)

Writes a list (simple list or proplist).

All the basic types (see Write) are suitable as items (integer, float, string, symbol, point, rect, list and void).

A list can contain other lists which can in turn contain other lists, etc.

**Parameters:**

**listName:** string. A name which identifies the written list. This name is required in order to read the list using readList(). Attention, the name is case sensitive ("aName" is considered different from "aname").

**theList:** the list to write. All Director's basic types are suitable. The list can be linear or a list of properties. The list can contain nested lists.

**Returns:**

The number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Note:**

A file can only contain one list per name. If you try to write a list, giving as its name, the name of a list previously written in the file, the previously written list will be replaced by the new one, i.e:

```
AbsObj.writeList("aName",[1,2,3])
```

```
Put AbsObj.readList("aName")
-- [1, 2, 3]
AbsObj.writeList(,"aName",[4,5,6])
Put AbsObj.readList("aName")
-- [4, 5, 6]
```

**Example:**

```
AbsObj=extra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.writeList( "ListName", [#name:"Andrew", #firstName:"Paul",
#age:42, #children:[[#firstName:"Julia", #age:12], [#firstName:"Joe",
#age:15]])
-- 193
AbsObj=VOID
```

---

## Pointer and file size functions

---

The functions return #err if an error occurs.

### Length()

Returns the size of the file in bytes.

**Returns:**

An integer if the size is less than or equal to 2147483647  
A float value if the size is greater than 2147483647  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Example:**

```
AbsObj=extra("AbsObj").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.length()
-- 178250
AbsObj=VOID
```

### GetPosition()

Returns the current position of the file pointer.

**Returns:**

An integer specifying the position of the pointer.  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Note:**

GetPosition does not return the correct position if the file is greater than 4 Gb. In this instance, an integer is returned which may be an incorrect value.

**Example:**

```
AbsObj=extra("AbsObj").new()
Put AbsObj.open("c:\file.dat",0,0)
-- #Ok
Put AbsObj.readByte()
-- 45
Put AbsObj.getPosition()
-- 4
AbsObj=VOID
```

## SetPosition(integer position, integer mode)

Positions the file pointer.

The pointer can be positioned beyond the end of the file without causing an error. If the file is open in read mode, an #eof will be returned when the read function is next used. If the file is open in write mode, its size will be increased up to this position when the next write operation occurs or when a call to the setEndOfFile() is made.

**Parameter:**

**Position:** an integer. Specifies pointer displacement in bytes.

**Mode:** an integer.

- 0 (FILE\_BEGIN), pointer displacement is calculated from the start of the file. Position is always  $\geq 0$
- 1 (FILE\_CURRENT), pointer displacement is calculated from the pointer's current position (position is positive or negative).
- 2 (FILE\_END), pointer displacement is calculated from the end of the file. Position can be positive or negative.

**Returns:**

An integer. The new absolute position of the pointer.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Note:**

It is not possible to set an absolute position greater than 2 Gb.

i.e. setPosition(2147483648,0) will not function correctly (2147483648 > 2Gb)

However, it is possible, by way of relative displacement, to move the pointer to a position greater than 2 Gb.

SetPosition(1073741824,0) – positions the pointer at 1 Gb

SetPosition(1073741824,1) – positions the pointer 1 Gb further

SetPosition(1073741824,1) – positions the pointer 1 Gb further again

The file pointer is now 3 Gb from the start of the file.

**Example:**

```

AbsObj=extra("AbsObj").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
-- Position the file pointer 20 bytes from the start of the file.
Put AbsObj.setPosition(20,0)
-- 20
-- Move the file pointer 30 bytes from its current position.
Put AbsObj.setPosition(30,1)
-- 30
AbsObj=VOID

```

## SetEndOfFile()

Sets the end of the file at the current position of the pointer. It is possible to truncate the file if the pointer is positioned before the end of the file, or enlarge the file if the pointer is positioned beyond the end of the file.

### Returns:

An integer if the new file size is less than or equal to 2147483647  
A float value if the new size is greater than 2147483648  
The symbol #err if an error occurs. Consult GetError() to obtain the error code.

### Note:

Windows 95 and 98 do not support files greater than 2 Gb in size.

### Example:

```

AbsObj=extra("AbsObj").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
AbsObj.SetPosition(0,0)
AbsObj.SetEndOfFile()           -- the file is reset to zero
AbsObj.SetPosition(1000, 0)
size = myFile.SetEndOfFile()   -- the file is enlarged by 1000 bytes
AbsObj=VOID

```

## General Functions

### SetMarker(string name)

Writes a marker in the file. These markers can then be used to locate values.

This function moves the file pointer by a number of bytes equal to 4+length of name.

These markers are used to define positions in a file, and to subsequently return to those positions using the gotoMarker() function.

**Parameter:**

**Name:** a string. The string which is used for the marker's name.

**Returns:**

An integer: the number of bytes written.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Note:**

The name passed in the parameter is case sensitive ("aName" is different from "aname").

**Example:**

```
AbsObj=extra("AbsObj").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.setMarker("aName")
-- 9
AbsObj=VOID
```

## GotoMarker(string name)

Moves the file pointer to the position just past the marker written using setMarker().

**Parameter:**

**Name:** a string. The name of the marker to find

**Returns:**

#ok if the marker is found and the pointer is correctly positioned.

The symbol #err if an error occurs. Consult GetError() to obtain the error code.

**Note:**

The name passed in the parameter is case sensitive ("aName" is different from "aname").

**Example:**

```
AbsObj=extra("AbsObj").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.gotoMarker("aName")
-- #Ok
AbsObj=VOID
```

## Flush()

Forces writing of all memory buffers to disk/physical media. Windows can delay writing of data to disk/physical media. This function forces data to be written.

**Returns:**

The symbol #ok if data in all buffers has been written correctly.  
The symbol #err if an error occurs. Consult GetLastError() to obtain the error code.

**Example:**

```
AbsObj=xtra("AbsObj").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.writeInt(133)
-- 4
Put AbsObj.flush()
-- #Ok
AbsObj=VOID
```

---

## Error Functions

---

### GetError()

Returns the code of the last error generated. This code corresponds to Windows error codes (0 = no error).

**Returns:**

An integer corresponding to the last error generated by a call to one of functions of the Absfile xtra. All the functions modify this code when executed (except for GetLastError()), even when the function executes correctly. In this case a value of 0 is returned.

**Example:**

```
AbsObj=xtra("AbsFile").new()
Put AbsObj.open("c:\file.dat",1,3)
-- #Ok
Put AbsObj.getError()
-- 0
AbsObj=VOID
```

### GetErrorText(integer codeErreur)

Returns a textual description of the error, corresponding to codeErreur.

**Parameter:**

CodeErreur: An integer corresponding to a Windows error code.

**Returns:**

A string corresponding to the text of the passed error code. The value returned is operating system dependent (version and language).

**Note:**

Error messages are dependent on the user's operating system and system language.

**Example:**

```
AbsObj=xtra("AbsFile").new()
Put AbsObj.getErrorText(2)
-- "The specified file cannot be found."
AbsObj=VOID
```

## Error Codes

The program ErrorInfo.exe is provided with the xtra. It enables you to obtain the error description for a given error code.

Run ErrorInfo.exe  
Enter the error number.  
Click on "Find"

**Codes for common errors:**

1	ERROR_INVALID_FUNCTION	Call to a function when no file is open.
2	ERROR_FILE_NOT_FOUND	The system cannot find the file specified.
3	ERROR_PATH_NOT_FOUND	The system cannot find the path specified.
5	ERROR_ACCESS_DENIED	Access denied. E.g. attempted write to a file opened in read-only mode or to a file on a CD-Rom, or attempted access to a file for which the user does not have the necessary permissions, or is an encrypted file, etc.
8	ERROR_NOT_ENOUGH_MEMORY	Not enough memory available. This can occur when reading very large amounts of data.
23	ERROR_CRC	Data read/write error due to disk/physical media problem.
32	ERROR_SHARING_VIOLATION	Sharing error
87	ERROR_INVALID_PARAMETER	Bad parameter, returned by Write or WriteList if a data type not managed by the function(s) was used.
2012	ERROR_TAG_NOT_FOUND	The marker or the list name could not be found.