Please note:

In this file you will find a description of some VB Script file system controls which are valid only for Windows CE systems and are different from those used for Windows-based systems.

File System Controls

This control consists of two controls, the **File** control and the **File System** control, that work together to provide basic file input and output functionality. The two controls also enable the manipulation of files and directories. In addition, the **FileSystem** control provides access to the **File** object through the **File** control and to the **FileSystem** object through the **FileSystem** control.

Library Name

FILECTLCtl

DLL Name

Mscefile.dll

The **File** control supports the following properties:

Attr	**Loc**	**Seek**
EOF	**LOF**	

The **File** control supports the following methods:

Close (File)	**InputFields**	**Put**
Get	**LineInputString**	**WriteFields**
Input	**LinePrint**	
InputB	**Open**	

The **FileSystem** control supports the following methods:

Dir	**GetAttr**	**RmDir**
FileCopy	**Kill**	**SetAttr**
FileDateTime	**MkDir**	
FileLen	**MoveFile**	

The **Function:**

CreateObject

Remarks

File System controls are unique to the Windows CE Toolkit for Visual Basic 6.0

CreateObject

This function creates a reference to an Automation object.

Syntax

CreateObject(*object*)

Parameters

object

A string containing the ProgID of the object to create.

Return Values

Returns a reference to an Automation object.

Remarks

Use **CreateObject** to create non-visible ActiveX controls at run time. You cannot use **CreateObject** to create graphical objects such as a **TreeView** control or a **ListView** control. **CreateObject** produces objects that cannot respond to events. To produce objects that can respond to events, use the **CreateObjectWithEvents** function. The following table lists the ProgIDs for the ActiveX controls without events.

Control	ProgID
Microsoft CE **File** control 6.0	.file
Microsoft CE **FileSystem** control 6.0	.filesystem
Microsoft CE **ImageList** control 6.0	CEimageList.imagelistctrl

```
Dim f, fwModeAppend
Set f = CreateObject("FileCtl.File")
fwModeAppend=8
f.Open "\Storage Card\testfile.txt", fwModeAppend
f.Close
```

MoveFile

This method renames an existing file or a directory, including all its subdirectories.

Syntax

filesystem.**MoveFile** *PathName*, *NewPathName*

Parameters

filesystem

Reference to a **FileSystem** control.

PathName

String that contains the file name.

NewPathName

String that contains the file name to copy to.

Return Value

None.

Put

This method writes data from a variable to a disk file.

Syntax

file.**Put** *data*, [*recnumber*]

Parameters

data

Required. **Variant** variable that contains data to be written to disk.

recnumber

Optional. **Variant** (**Long**). Record number (Random mode files) or byte number (Binary mode files) at which writing begins.

Return Value

None.

Remarks

Data written with **Put** usually is read from a file with **Get**.
The first record or byte in a file is at position 1, the second record or byte is at position 2, and so on. If you omit *recnumber*, the next record or byte after the last **Get** or **Put** method or pointed to by the last **Seek** function is written.

For files opened in Random mode, the following rules apply:

- If the length of the data being written is less than the length specified in the **Len** clause of the **Open** method, **Put** writes subsequent records on record-length boundaries. The space between the end of one record and the beginning of the next record is padded with the existing contents of the file buffer. Because the amount of padding data cannot be determined with any certainty, it generally is a good idea to have the record length match the length of the data being written. If the length of the data being written is greater than the length specified in the Len clause of the **Open** method, an error occurs.
- If the variable being written is a **Variant** of a numeric type, **Put** writes 2 bytes identifying the **VarType** of the **Variant** and then writes the variable. For example, when writing a **Variant** of **VarType 3**, **Put** writes 6 bytes: 2 bytes identifying the **Variant** as **VarType 3** (**Long**) and 4 bytes containing the **Long** data. The record length specified by the **Len** clause in the **Open** method must be at least 2 bytes greater than the actual number of bytes required to store the variable.

You can use the **Put** method to write a **Variant** array to disk, but you cannot use **Put** to write a scalar **Variant** containing an array to disk. You also cannot use **Put** to write objects to disk.
If the variable being written is a **Variant** of VarType 8 (String), **Put** writes 2 bytes identifying the **VarType** and 2 bytes indicating the length of the string. It then writes the string data. The record length specified by the **Len** clause in the **Open** method must be at least 4 bytes greater than the actual length of the string.
If the variable being written is a dynamic array, **Put** writes a descriptor whose length equals 2 plus 8 times the number of dimensions, that is, 2 + 8 * NumberOfDimensions. The record length specified by the Len clause in the **Open** method must be greater than or equal to the sum of all the bytes required to write the array data and the array descriptor. For example, the following array declaration requires 118 bytes when the array is written to disk.
For files opened in Binary mode, the **Len** clause in the **Open** method has no effect. **Put** writes all variables to disk contiguously; that is, with no padding between records.

LinePrint

This method writes a single line to an open sequential file.

Syntax

file.**LinePrint** *output*

Parameters

file

Reference to a **File** control.

output

String expression to write to a file.

Return Value

None.

Remarks

Data written with **LinePrint** is usually read from a file with **LineInputString**.
A carriage return/line feed (Chr(13) + Chr(10)) sequence is appended to the end of the string.

Get

This method reads data from an open disk file into a variable.

Syntax

file.**Get** *Data*, [*Recnumber*]

Parameters

file

Reference to a **File** control.

Data

Required. **Variant** variable into which data is read.

Recnumber

Optional. **Variant**. Record number at which reading begins. For files opened in binary mode, *Recnumber* specifies the byte position.

Return Value

None.

Remarks

Data read with the **Get** method usually is written to a file with the **Put** method.
The first record or byte in a file is at position 1, the second record or byte is at position 2, and so on. If you omit *Recnumber*, the next record or byte following the last **Get** or **Put** method (or pointed to by the last **Seek** function) is read.

For files opened in Random mode, the following rules apply:

- If the length of the data being read is less than the length specified in the **Len** clause of the **Open** method, **Get** reads subsequent records on record-length boundaries. The space between the end of one record and the beginning of the next record is padded with the existing contents of the file buffer. Because the amount of padding data cannot be determined with any certainty, it is generally advisable to match the record length with the length of the data being read.
- If *Data* is a **Variant** of numeric type, **Get** reads 2 bytes identifying the **VarType** of the **Variant** and then reads the data that goes into the variable. For example, when reading a **Variant** of VarType 3, **Get** reads 6 bytes: 2 bytes identifying the **Variant** as VarType 3 (**Long**) and 4 bytes containing the **Long** data. The record length specified by the **Len** clause in the **Open** method must be at least 2 bytes greater than the actual number of bytes required to store the variable.
- You can use the **Get** method to read a **Variant** array from a disk, but you cannot use **Get** to read a scalar **Variant** containing an array. You also cannot use **Get** to read objects from a disk.
- If the variable being read into is a **Variant** of VarType 8 (**String**), **Get** reads 2 bytes identifying the VarType and 2 bytes indicating the length of the string. Then it reads the string data. The record length specified by the **Len** clause in the **Open** method must be at least 4 bytes greater than the actual length of the string.
- If the variable being read into is a dynamic array, **Get** reads a descriptor whose length equals 2 plus 8 times the number of dimensions, that is, 2 + 8 * NumberOfDimensions. The record length specified by the **Len** clause in the **Open** method must be greater than or equal to the sum of all the bytes required to read the array data and the array descriptor.

For files opened in Binary mode, the **Len** clause in the **Open** method has no effect. **Get** reads all variables from a disk contiguously; that is, with no padding between records.

LineInputString

This method reads a single line from an open sequential file and assigns it to a string variable.

Syntax

file.**LineInputString**

Parameters

file

Reference to a **File** control.

Return Value

None.

Remarks

Data read with **LineInputString** usually is written from a file with **LinePrint**.
The **LineInputString** method reads from a file one character at a time until it encounters a carriage return (Chr(13)) or carriage return/line feed (Chr(13) + Chr(10)) sequence. Carriage return/line feed sequences are skipped rather than appended to the character string.

FileCopy

This method copies an existing file to a new file.

Syntax

filesystem.**FileCopy** *PathName*, *NewPathName*

Parameters

filesystem

Reference to a **FileSystem** object.

PathName

String that contains the path and file name.

NewPathName

String that contains the file name and path of the new file.

Return Value

None.

Remarks

FileCopy returns an error if the new file does not exist.

Kill

This method deletes files from a disk.

Syntax

filesystem.**Kill** *pathname*

Parameters

filesystem

Reference to a **FileSystem** control.

pathname

Required. String expression that specifies one or more file names to be deleted. The *pathname* can include the directory or folder.

Return Value

None.

Remarks

The **Kill** method supports the use of multiple-character (*) and single-character (?) wildcards to specify multiple files.
An error occurs if you try to use **Kill** to delete an open file.

MkDir

This method creates a new directory.

Syntax

filesystem.**MkDir** *PathName*

Parameters

filesystem

Reference to a **FileSystem** control.

Pathname

String expression that contains the directory name.

Return Value

None.

Remarks

MkDir generates an error if the directory already exists.

Open

This method opens a file in either the Input (1), Output (2), Random (4), Append (8), or Binary mode (32).

Syntax

file.**Open** *pathname*, *mode,* [*access*], [*lock*], [*reclength*]

Parameters

file

Reference to a **File** control.

pathname

String expression that specifies a file name.

mode

Specifies the file mode: Input (1), Output (2), Random (4) , Append (8), or Binary (32).

access

Operation permitted on the open file: Read, Write, or ReadWrite [Default]. (1, 2, 3)

lock

Operations permitted on the open file by other processes: Shared, LockRead, LockWrite [Default], and LockReadWrite. (1, 2, 3, 0)

reclength

Number, in bytes, that is less than 32,767. For files opened for random access, this value is the record length. For sequential files, this value is the number of characters buffered.

Return Value

None.

Remarks

The *reclength* parameter is ignored if the mode is Binary. When opening a file in Random mode, you must specifiy a record size of greater than zero or an error will occur.

Attr

This property of the **File** control returns a number indicating the file mode that was used to open the file.

Syntax

file.**Attr**

Parameters

file

Reference to a **File** control.

Return Values

The return values listed in the following table indicate the file access mode. If the return value is 0, the file is closed.

Constant	Value
None	0
fsModeInput	1
fsModeOutput	2
fsModeRandom	4
fsModeAppend	8
fsModeBinary	32

Remarks

The **Attr** property is read-only. Use the **Open** method of the **File** control to set the file mode.

EOF

This property returns **True** when the end of a file opened for random or sequential input is reached.

Syntax

file.**EOF**

Parameters

file

Reference to a **File** control.

Remarks

Use the **EOF** property to avoid the error generated by attempting to read past the end of a file.
The **EOF** property returns **False** until the end of the file has been reached. For files opened with a **fsModeRandom** or **fsModeBinary** file mode, **EOF** returns **False** until the last executed **Get** statement is unable to read an entire record.
For files opened with a **fsModeBinary** file mode, an attempt to read through the file using the **Input** function until **EOF** returns **True** generates an error. Use the **LOF** and **LOC** properties instead of **EOF** when reading binary files with **Input**, or use **Get** when using the **EOF** property. For files opened with a **fsModeOutput** file mode, **EOF** always returns **True**.

Loc

This property returns a number specifying the current read/write position.

Syntax

file.**Loc**

Parameters

file

Reference to a **File** control.

Remarks

For files opened with the **fsModeRandom** file mode, **Loc** returns the number of the last record read or written. For files opened with all other modes, **Loc** returns the position of the last byte read or written.

LOF

This property returns a number representing the size, in bytes, of a file.

Syntax

file.**LOF**

Parameters

file

Reference to a **File** control.

Remarks

The **LOF** property can be used with the **Loc** property to guarantee that a read operation does not continue past the end of a file.

Seek

This property returns and sets the next position in a file that will be read or written.

Syntax

file.**Seek** [= *position*]

Parameters

file

Reference to a **File** control.

position

Numeric expression that specifies a position within a file.

Remarks

The **Seek** property specifies the next file position, whereas the **Loc** property specifies the current position. **Seek** always will be one more than **Loc**, except when a file is first opened and **Seek** and **Loc** are both 1.
Negative **Seek** or 0 causes an error.

Close (File)

This method closes an open **File** control.

Syntax

file.**Close**

Parameters

file

Name of a **File** control.

Return Value

None.

Remarks

Use the **Open** method to open a file.

Input

This method returns a string containing characters from a file opened in Input or Binary mode.

Syntax

file.**Input**(*number*)

Parameters

file

Reference to a **File** control.

number

Any valid numeric expression that specifies the number of characters to return.

Return Value

String containing characters read from *file*.

Remarks

Data read with the **Input** method usually is written to a file with the **LinePrint** or **Put** functions. Use this method only with files opened in Input or Binary mode.

Unlike the **LineInputString** method, the **Input** method returns all the characters it reads, including commas, carriage returns, line feeds, quotation marks, and leading spaces.

With files opened for Binary access, an attempt to read through the file using the **Input** method until the **EOF** function returns **True** generates an error. To avoid an error, use the **LOF** and **Loc** functions instead of **EOF** when reading binary files with the **Input** method or use **Get** when using the **EOF** function.

InputB

This method returns bytes from a file opened in Input or Binary mode.

Syntax

file.**InputB**(*number*)

Parameters

file

Reference to a **File** control.

number

Any valid numeric expression that specifies the number of bytes to return.

Return Value

Array containing bytes read from *file*.

Remarks

Data read with the **InputB** method usually is written to a file with the **LinePrint** or **Put** functions. Use this method only with files opened in Input or Binary mode.

InputFields

This method reads data from an open sequential file and returns a single dimension **Variant** array.

Syntax

file.**InputFields**(*number*)

Parameters

file

Reference to a **File** control.

number

Number of comma-delimited fields to read from the file.

Return Value

Array containing the fields read from the file.

Remarks

Data read with the **InputFields** method usually is written to a file with **WriteFields**. Use this method only with files opened in Input or Binary mode.
InputFields reads standard string or numeric data without modification. The following table shows how **InputFields** reads other input data.

Data	Value Assigned to Variable
Delimiting comma or blank line	Empty
#NULL#	Null
#TRUE# or #FALSE#	True or False
#yyyy-mm-dd hh:mm:ss#	The date and/or time represented by the expression

Double quotation marks ("") within input data are discarded.
If you reach the end of the file while you are inputting a data item, the input is terminated and an error occurs.

To correctly read data from a file into variables using **InputFields**, use the **WriteFields** method instead of the **LinePrint** method to write the data to the files. Using **WriteFields** ensures each separate data field is properly delimited.

WriteFields

This method writes data to a sequential file.

Syntax

file.**WriteFields** [d*ata*]

Parameters

file

Reference to a **File** control.

data

Variant or **Variant** array of numeric or string expressions to write to a file.

Return Value

None.

Remarks

Data written with **WriteFields** is usually read from a file with **InputFields**.
If you omit d*ata*, a blank line is printed to the file.

When **WriteFields** is used to write data to a file, several universal assumptions are followed so that the data can always be read and correctly interpreted using **InputFields**, regardless of locale:

- Numeric data is always written using the period as the decimal separator.
- For Boolean data, either #TRUE# or #FALSE# is printed. The **True** and **False** keywords are not translated, regardless of locale.
- Date data is written to the file using the universal date format. When either the date or the time component is missing or is zero, only the component provided gets written to the file.
- Nothing is written to the file if *Data* is **Empty**. However, for **Null** data, #NULL# is written.
- If d*ata* is **Null**, #NULL# is written to the file.

The **WriteFields** method inserts commas between items and quotation marks around strings as they are written to the file. You do not have to put explicit delimiters in the list. **WriteFields** inserts a newline character—that is, a carriage return/line feed (Chr(13) + Chr(10))—after it has written the final character in *data* to the file.

Dir

This method returns the name of a file, directory, or folder that matches a specified pattern or file attribute.

Syntax

file.**Dir**(*pathname*,[*attributes*])

Parameters

file

Reference to a **FileSystem** control.

pathname

Optional. String expression that specifies a file name or path.

attributes

Optional. Numeric expression whose sum specifies file attributes.
If omitted, all files that match *pathname* are returned.

The following table describes the parameter settings of *attributes*.

Constant	Value	Description
fsAttrNormal	0	Normal
fsAttrReadOnly	1	Read-only
fsAttrHidden	2	Hidden
fsAttrSystem	4	System file
fsAttrVolume	8	Volume label. If specified, all other attributes are ignored.
fsAttrDirectory	16	Directory or folder
fsAttrArchive	32	Archive

Return Value

String. File name that matches *pathname* and *attributes*. **Dir** returns a zero-length string ("") if *pathname* is not found.

Remarks

Dir supports the use of multiple-character (*) and single-character (?) wildcards to specify multiple files. You must specify *pathname* the first time you call the **Dir** method. In addition, if you specify file attributes you must include *pathname*.
The **Dir** method returns the first file name that matches *pathname*. To get any additional file names that match *pathname*, call **Dir** again with no parameters. When no more file names match, **Dir** returns a zero-length string (" "). Once a zero-length string is returned, you must specify *pathname* in subsequent calls.

FileDateTime

This method returns a variant (**Date**) that indicates the date and time when a file was created or last modified.

Syntax

filesystem.**FileDateTime**(*pathname*)

Parameters

filesystem

Reference to a **FileSystem** control.

pathname

Required. String expression that specifies a file name. The *pathname* can include a directory or folder.

Return Value

Returns the date the file was last modified.

Remarks

FileDateTime returns an error if the new file does not exist.

FileLen

This method returns a value specifying the length, in bytes, of a file.

Syntax

filesystem.**FileLen**(*pathname*)

Parameters

filesystem

Reference to a **FileSystem** control.

pathname

Required. String expression that specifies a file. The *pathname* can include a directory or folder.

Return Value

Returns the number of bytes in a file.

Remarks

If the specified file is open when the **FileLen** method is called, the value returned represents the size of the file immediately before it was opened.

GetAttr

This method returns a number representing the attributes of a file, directory, or folder.

Syntax

filesystem.**GetAttr**(*pathname*)

Parameters

filesystem

Reference to a **FileSystem** control.

pathname

Required. String expression that specifies a file name or directory or a folder name. The *pathname* can include the directory or folder.

Return Value

Sum of attribute values. The following table shows the sums that can be returned.

Constant	Value	Description
vbNormal	0	Normal
VbReadOnly	1	Read-only
VbHidden	2	Hidden
VbSystem	4	System
VbDirectory	16	Directory or folder
VbArchive	32	File has changed since last backup

Remarks

To determine which attributes are set, use the **And** operator to perform a bitwise comparison of the value returned by the **GetAttr** method and the value of the individual file attribute you want. If the result is not zero, that attribute is set for the named file.

RmDir

This method deletes an existing empty directory.

Syntax

filesystem.**RmDir** *PathName*

Parameters

filesystem

Reference to a **FileSystem** control.

PathName

String that contains the directory name.

Return Value

None.

Remarks

The directory must be empty before it can be removed. You must specify a complete file path.

SetAttr

This method sets attribute data for a file.

Syntax

filesystem.**SetAttr** *pathname*, *attributes*

Parameters

filesystem

Reference to a **FileSystem** control.

pathname

Required. String expression that specifies a file name. The file name can include a path.

attributes

Required. Numeric expression whose sum specifies file attributes. The following table shows the parameter settings of attributes.

Constant	Value	Description
vbNormal	0	Normal (default)
vbReadOnly	1	Read-only
vbHidden	2	Hidden
VbSystem	4	System file
VbArchive	32	File has changed since last backup

Return Value

None.

Remarks

A run-time error occurs if you try to set the attributes of an open file.