

## Corel SCRIPT Editor commands and functions

[AddLineAfter command/function](#)

[AddLineBefore command/function](#)

[CheckSyntax command](#)

[DeleteLine command/function](#)

[Execute command](#)

[FileClose command/function](#)

[FileNew command](#)

[FileOpen command/function](#)

[FileSave command/function](#)

[FileSaveAs command/function](#)

[GetColumnNumber function](#)

[GetLineNumber function](#)

[GetLineText function](#)

[GoToColumn command/function](#)

[GoToEndOfDoc command/function](#)

[GoToLine command/function](#)

[MakeCSB command/function](#)

[MakeDLL command/function](#)

[MakeEXE command/function](#)

[MoveLineDown command/function](#)

[MoveLineUp command/function](#)

[ReplaceLine command/function](#)

[Run command](#)

[SetVisible command](#)

## **.FileClose (Corel SCRIPT Editor)**

Command: .FileClose

Function: ReturnValue = .FileClose ( )

### **Description**

Closes the active script.

### **Return Value**

The .FileClose function returns one of the following values:

- TRUE (-1) — the active script was closed
- FALSE (0) — the active script was not closed

### **Notes**

- The .FileClose command must be preceded by a [.FileNew](#) or [.FileOpen](#) command or else an error occurs. The error occurs because the .FileClose command cannot close the script which is executing the command.
- If you do not precede the .FileClose command by either the [.FileSave](#) or [.FileSaveAs](#) command, you will lose the changes to the script.
- The .FileClose command corresponds to the Close command in the File menu in the Corel SCRIPT Editor. Click File, Close.

### **Example**

The following example opens the Graphics.CSC script , executes it, and closes it:

```
.FileOpen "C:\MyScripts\Graphics.CSC"  
.Execute  
.FileClose
```

---

[{button.AL\('cse\\_file\\_cse;;;;0."Defaultoverview".\)} Related Topics](#)

## **.FileNew (Corel SCRIPT Editor)**

.FileNew

### **Description**

Creates a new Corel SCRIPT script. The new script becomes the active script and the insertion point is placed at the beginning of the first line.

### **Note**

- The .FileNew command corresponds to the New command in the File menu in the Corel SCRIPT Editor. Click File, New.

### **Example**

The following example creates a new Corel SCRIPT script:

```
.FileNew
```

---

{button ,AL(^cse\_file\_cse;;;;',0,"Defaultoverview",,)} Related Topics

## .FileOpen (Corel SCRIPT Editor)

Command: .FileOpen FileName

Function: ReturnValue = .FileOpen (FileName)

### Description

Opens a saved Corel SCRIPT script. The opened script becomes the active script and the insertion point is placed at the beginning of the first line.

### Return Value

The .FileOpen function returns one of the following values:

- TRUE (-1) — the active script was opened
- FALSE (0) — the active script was not opened

Parameter	Description
FileName	<u>String expression</u> that specifies the name and path of the script to open.

### Note

- The .FileOpen command corresponds to the Open command in the File menu in the Corel SCRIPT Editor. Click File, Open.

### Example

The following example opens the Graphics.CSC script, executes it, and closes it.

```
.FileOpen "C:\MyScripts\Graphics.CSC"  
.Execute  
.FileClose
```

You can also specify the first line as follows:

```
ReturnValue = .FileOpen ("C:\MyScripts\Graphics.CSC")
```

---

[button.AL\(^cse\\_file\\_cse:::;0,"Defaultoverview".\)\) Related Topics](#)

## **.FileSave (Corel SCRIPT Editor)**

Command: .FileSave

Function: ReturnValue = .FileSave ( )

### **Description**

Saves the active script using its current name and location.

### **Return Value**

The .FileSave function returns one of the following values:

- TRUE (-1) — the active script was saved
- FALSE (0) — the active script was not saved

### **Note**

- The .FileSave command corresponds to the Save command in the File menu in the Corel SCRIPT Editor. Click File, Save.
- If the script is not named, the Save As dialog box opens.
- If the .FileSave command is not preceded by a [.FileNew](#) or [.FileOpen](#) command, Corel SCRIPT tries to save the script that is executing the command.

### **Example**

The following example saves the active script:

```
.FileSave
```

You can also specify the line as follows:

```
ReturnValue = .FileSave ( )
```

---

[{button.AL\(^cse\\_file\\_cse;;;;;0,"Defaultoverview",\)} Related Topics](#)

## .FileSaveAs (Corel SCRIPT Editor)

Command: .FileSaveAs FileName

Function: ReturnValue = . .FileSaveAs (FileName)

### Description

Saves the active script under a new name or in a different location.

### Return Value

The .FileSaveAs function returns one of the following values:

- TRUE (-1) — the active script was saved
- FALSE (0) — the active script was not saved

Parameter	Description
FileName	<u>String expression</u> that specifies the name and path to save the script under.

### Note

- The .FileSaveAs command corresponds to the Save As command in the File menu in the Corel SCRIPT Editor. Click File, Save As.

### Example

The following example opens the Graphics.CSC script, saves it on the D drive, and closes it:

```
.FileOpen "C:\MyScripts\Graphics.CSC"  
.FileSaveAs "D:\Graphics.CSC"  
.FileClose
```

You can also specify the second line as follows:

```
ReturnValue = .FileSaveAs ("D:\Graphics.CSC")
```

---

[button.AL\(^cse\\_file\\_cse:::'.0,"Defaultoverview".\)\)](#) Related Topics



## **.CheckSyntax (Corel SCRIPT Editor)**

.CheckSyntax

### **Description**

Checks for syntax errors in the active script. Common syntax errors include misspelling commands, missing operators, and missing punctuation. If errors are found, error messages appear in the Compiler Output window.

### **Note**

- The .CheckSyntax command corresponds to the Check Syntax command in the Debug menu in the Corel SCRIPT Editor. Click Debug, Check Syntax.

### **Example**

.CheckSyntax

---

{button.AL(^cse\_debug\_cse;;;;;.0,"Defaultoverview",)} Related Topics



## **.Execute (Corel SCRIPT Editor)**

.Execute

### **Description**

Runs the active script. The .Execute command ignores all debugging information, including script breakpoints during script execution. To debug a script, use the .Run command. Running a script in debug mode is noticeably slower than running a script using the .Execute command.

### **Note**

- The .Execute command corresponds to the Execute command in the Debug menu in the Corel SCRIPT Editor. Click Debug, Execute.

### **Example**

The following example opens the Graphics.CSC script , executes it, and closes it.

```
.FileOpen "C:\MyScripts\Graphics.CSC"  
.Execute  
.FileClose
```

---

[{button.AL\(^cse\\_debug\\_cse;::::';0,"Defaultoverview",\)} Related Topics](#)

## **.Run (Corel SCRIPT Editor)**

.Run

### **Description**

Runs the active script in debug mode. Script execution stops at breakpoints, or when the end of the script is reached. Running a script in debug mode is noticeably slower than running a script using the [.Execute](#) command.

### **Note**

- The .Run command corresponds to the Run command in the Debug menu in the Corel SCRIPT Editor. Click Debug, Run.

### **Example**

The following example opens the Graphics.CSC script , runs it, and closes it.

```
.FileOpen "C:\MyScripts\Graphics.CSC"  
.Run  
.FileClose
```

---

[{button.AL\(^cse\\_debug\\_cse;;;;;0,"Defaultoverview",\)} Related Topics](#)



## **.AddLineAfter (Corel SCRIPT Editor)**

Command: .AddLineAfter Text

Function: ReturnValue = .AddLineAfter (Text)

### **Description**

Inserts a line and text after the line containing the insertion point.

### **Return Value**

The .AddLineAfter function returns one of the following values:

- TRUE (-1) — the line and text were added
- FALSE (0) — the line and text were not added

Parameter	Description
Text	<u>String expression</u> that specifies the text to add to the inserted line.

### **Example**

The following example adds a comment line after the line containing the insertion point:

```
.AddLineAfter "REM This is a comment to be added"
```

You can also specify the line as follows:

```
ReturnValue = .AddLineAfter ("REM This is a comment to be added")
```

---

[{button.AL\(^cse\\_edit\\_commands;:::::0,"Defaulttooverview".\)} Related Topics](#)

## **.AddLineBefore (Corel SCRIPT Editor)**

Command: .AddLineBefore Text

Function: ReturnValue = .AddLineBefore (Text)

### **Description**

Inserts a line and text before the line containing the insertion point.

### **Return Value**

The .AddLineBefore function returns one of the following values:

- TRUE (-1) — the line and text were added
- FALSE (0) — the line and text were not added

Parameter	Description
Text	<u>String expression</u> that specifies the text to add to the inserted line.

### **Example**

The following example adds a comment line before the line containing the insertion point.

```
.AddLineBefore "REM This is a comment to be added"
```

You can also specify the line as follows:

```
ReturnValue = .AddLineBefore ("REM This is a comment to be added")
```

---

[{button.AL\(^cse\\_edit\\_commands;:::::0,"Defaulttooverview".\)} Related Topics](#)

## **.DeleteLine (Corel SCRIPT Editor)**

Command: .DeleteLine

Function: ReturnValue = .DeleteLine ( )

### **Description**

Deletes the line containing the insertion point. The insertion point is then placed in the line that follows the deleted line.

### **Return Value**

The .DeleteLine function returns one of the following values:

- TRUE (-1) — the line was deleted
- FALSE (0) — the line was not deleted

### **Example**

The following example deletes the line containing the insertion point:

```
.DeleteLine
```

You can also specify the line as follows:

```
ReturnValue = .DeleteLine ( )
```

---

[{button.AL\(^cse\\_edit\\_commands;::::'0,"Defaultoverview",,\)} Related Topics](#)

## **.GetLineText (Corel SCRIPT Editor)**

```
ReturnValue$ = .GetLineText ( )
```

### **Description**

Returns the text from the line containing the insertion point.

### **Return Value**

The string variable that is assigned the text in the line containing the insertion point

### **Example**

The following example passes the text in the line containing the insertion point to the RV\_Text variable. This text is then displayed in a message box.

```
RV_Text$ = .GetLineText ( )  
MESSAGE RV_Text$
```

---

[{button .AL\(^cse\\_edit\\_commands;::::'0."Defaultoverview",\)} Related Topics](#)

## **.ReplaceLine (Corel SCRIPT Editor)**

Command: .ReplaceLine Text

Function: ReturnValue = .ReplaceLine (Text)

### **Description**

Replaces the text in the line containing the insertion point with specified text.

### **Return Value**

The .ReplaceLine function returns one of the following values:

- TRUE (-1) — the text was inserted
- FALSE (0) — the text was not inserted

Parameter	Description
Text	<u>String expression</u> that specifies the text to insert in the line containing the insertion point.

### **Example**

The following example replaces the line containing the insertion point with a comment line.

```
.ReplaceLine "REM This is a comment to be added"
```

You can also specify the line as follows:

```
ReturnValue = .ReplaceLine ("REM This is a comment to be added")
```

---

[{button.AL\(^cse\\_edit\\_commands;:::::0,"Defaulttooverview".\)} Related Topics](#)





## **.GetColumnNumber (Corel SCRIPT Editor)**

ReturnValue& = .GetColumnNumber ( )

### **Description**

Returns the column number position of the insertion point.

### **Return Value**

The numeric variable that is assigned the column number position of the insertion point.

### **Example**

The following example assigns the column number positions to RV\_Number:

```
RV_Number& = .GetColumnNumber ( )
```

---

{button.AL(^cse\_nav\_cse;;;;0."Defaultoverview".)} Related Topics

## **.GetLineNumber (Corel SCRIPT Editor)**

ReturnValue& = .GetLineNumber ( )

### **Description**

Returns the line number of the line containing the insertion point.

### **Return Value**

The numeric variable that is assigned the line number of the line containing the insertion point.

### **Example**

The following example sends the insertion point to the last line in the active script. The line number is assigned to RV\_Number. The last line displays a message box indicating the number of line in the script.

```
.GoToEndOfDoc  
RV_Number& = .GetLineNumber ( )  
MESSAGE RV_Number& & " are in the active script"
```

---

[{button.AL\(^cse\\_nav\\_cse;;;;;0."Defaultoverview".\)} Related Topics](#)

## **.GoToColumn (Corel SCRIPT Editor)**

Command: .GoToColumn ColumnNumber

Function: ReturnValue = .GoToColumn (ColumnNumber)

### **Description**

Sends the insertion point to a specified column in the line containing the insertion point.

### **Return Value**

The .GoToColumn function returns one of the following values:

- TRUE (-1) — the insertion point was sent to the specified column
- FALSE (0) — the insertion point was not sent to the specified column

Parameter	Description
ColumnNumber	Lets you specify the column to move the insertion point to.

### **Note**

- If the column does not exist in the active script, the insertion point is not moved.

### **Example**

The following example sends the insertion point to the fourteenth column in the line containing the insertion point:

```
.GoToColumn 14
```

You can also specify the column as follows:

```
ReturnValue = .GoToColumn (14)
```

---

[{button .AL\(^cse\\_nav\\_cse;;;;;0,"Defaultoverview",\)} Related Topics](#)

## **.GoToEndOfDoc (Corel SCRIPT Editor)**

Command: .GoToEndOfDoc

Function: ReturnValue = .GoToEndOfDoc ( )

### **Description**

Sends the insertion point to the beginning of the last line in the active script.

### **Return Value**

The .GoToEndOfDoc function returns one of the following variables:

- TRUE (-1) — the insertion point was sent to the last line
- FALSE (0) — the insertion point was not sent to the last line

### **Example**

The following example sends the insertion point to the last line in the active script. The line number is assigned to RV\_Number. The last line displays a message box indicating the number of the line in the script.

```
.GoToEndOfDoc
RV_Number& = .GetLineNumber ( )
MESSAGE RV_Number& & " are in the active script"
```

You can also specify the first line as follows:

```
ReturnValue = .GoToEndOfDoc ( )
```

---

[{button.AL\(^cse\\_nav\\_cse:;;;;0."Defaultoverview".\)} Related Topics](#)

## **.GoToLine (Corel SCRIPT Editor)**

Command: .GoToLine LineNumber

Function: ReturnValue = .GoToLine (LineNumber)

### **Description**

Sends the insertion point to the beginning of a specified line.

### **Return Value**

The .GoToLine function returns one of the following variables:

- TRUE (-1) — the insertion point was sent to the specified line
- FALSE (0) — the insertion point was not sent to the specified line

Parameter	Description
LineNumber	Lets you specify the line to move the insertion point to in the active script.

### **Notes**

- The .GoToLine command corresponds to the Go To dialog box. Click Search, Go To Line.
- If the line does not exist in the active script, the insertion point is not moved.

### **Example**

The following example sends the insertion point to the fourteenth line in the active script.

```
.GoToLine 14
```

You can also specify the line as follows:

```
ReturnValue = .GoToLine (14)
```

---

[{button.AL\(^cse\\_nav\\_cse;;;;;0,"Defaultoverview",\)} Related Topics](#)

## **.MoveLineDown (Corel SCRIPT Editor)**

Command: .MoveLineDown

Function: ReturnValue = .MoveLineDown ( )

### **Description**

Moves the insertion point to the line after the line containing the insertion point. The insertion point is placed at the beginning of the line.

### **Return Value**

The .MoveLineDown function returns one of the following values:

- TRUE (-1) — the insertion point was moved
- FALSE (0) — the insertion point was not moved

### **Notes**

- The insertion point is not moved if it is in the first line of the script.

### **Example**

The following example moves the insertion point down one line in a script:

```
.MoveLineDown
```

You can also specify the line as follows:

```
ReturnValue = .MoveLineDown ( )
```

---

[button.AL\('cse\\_nav\\_cse:::0,"Defaultoverview"\). Related Topics](#)

## **.MoveLineUp (Corel SCRIPT Editor)**

Command: .MoveLineUp

Function: ReturnValue = .MoveLineUp ( )

### **Description**

Moves the insertion point to the line before the line containing the insertion point. The insertion point is placed at the beginning of the line.

### **Return Value**

The .MoveLineUp function returns one of the following values:

- TRUE (-1) — the insertion point was moved
- FALSE (0) — the insertion point was not moved

### **Note**

- The insertion point is not moved if it is in the first line of the script.

### **Example**

The following example moves the insertion point up one line in a script:

```
.MoveLineUp
```

You can also specify the line can also be specified as follows:

```
ReturnValue = .MoveLineUp ( )
```

---

[button.AL\('cse\\_nav\\_cse;;;;0,"Defaultoverview"\).Related Topics](#)



## .MakeCAO (Corel SCRIPT Editor)

Command: .MakeCAO CAOName

Function: ReturnValue = .MakeCAO (CAOName)

### Description

Creates a Corel Add-on file.

### Return Value

The .MakeCAO function returns one of the following values:

- TRUE (-1) — the CAO was created
- FALSE (0) — the CAO was not created

Parameter	Description
CAOName	<u>String expression</u> that specifies the CAO name and path. If the CAO already exists, it will be overwritten. If the path is not specified, the CAO is saved in the active folder. You should provide a CAO extension with the CAO name.

### Note

- The .MakeCAO command corresponds to the Make Corel Add-on dialog box in the Corel SCRIPT Editor. Click File, Make CAO.

### Example

The following example creates a CAO named TOOLS:

```
.MakeCAO "C:\Myfiles\tools.CAO"
```

You can also specify the line as follows:

```
ReturnValue = .MakeCAO ("C:\Myfiles\tools.CAO")
```

---

[{button.AL\(C:\cse\\_make\;";0."Defaultoverview".\)} Related Topics](#)

## .MakeCSB (Corel SCRIPT Editor)

Command: .MakeCSB CSBName

Function: ReturnValue = .MakeCSB (CSBName)

### Description

Creates a Corel SCRIPT Binary file. A Binary file is a compiled version of a script file.

### Return Value

The .MakeCSB function returns one of the following values:

- TRUE (-1) — the CSB was created
- FALSE (0) — the CSB was not created

Parameter	Description
CSBName	<u>String expression</u> that specifies the CSB name and path. If the CSB already exists, it will be overwritten. If the path is not specified, the CSB is saved in the active folder. You should provide a CSB extension with the CSB name.

### Note

- The .MakeCSB command corresponds to the Make Corel SCRIPT Binary dialog box in the Corel SCRIPT Editor. Click File, Make CSB.

### Example

The following example creates a CSB named PRINTERS:

```
.MakeCSB "C:\Myfiles\printers.CSB"
```

You can also specify the line as follows:

```
ReturnValue = .MakeCSB ("C:\Myfiles\printers.CSB")
```

---

[button.AL\(cse\\_make;::::'.0."Defaultoverview".\)} Related Topics](#)

## .MakeDLL (Corel SCRIPT Editor)

Command: .MakeDLL DLLName

Function: ReturnValue = .MakeDLL (DLLName)

### Description

Creates a dynamic link library (DLL) from the active script. DLLs contain a library of functions that can be loaded by Corel SCRIPT or other applications at run time.

### Return Value

The .MakeDLL function returns one of the following values:

- TRUE (-1) — the DLL was created
- FALSE (0) — the DLL was not created

Parameter	Description
DLLName	<u>String expression</u> that specifies the DLL name and path. If the DLL already exists, it will be overwritten. If the path is not specified, the DLL is saved in the active folder. You should provide a DLL extension with the DLL name.

### Note

- The .MakeDLL command corresponds to the Make DLL File dialog box in the Corel SCRIPT Editor. Click File, Make DLL.

### Example

The following example creates a DLL named FINANCE:

```
.MakeDLL "C:\Myfiles\finance.DLL"
```

You can also specify the line as follows:

```
ReturnValue = .MakeDLL ("C:\Myfiles\finance.DLL")
```

---

[button.AL\(cse\\_make,,,,,0,"Defaultoverview"\).}\) Related Topics](#)

## .MakeEXE (Corel SCRIPT Editor)

Command: .MakeEXE EXENAME

Function: ReturnValue = .MakeEXE (EXENAME)

### Description

Creates a Corel SCRIPT executable from the active script. An executable is an application that runs without opening or starting the Corel SCRIPT Editor, or any other Corel application.

### Return Value

The .MakeEXE function returns one of the following values:

- TRUE (-1) — the executable was created
- FALSE (0) — the executable was not created

Parameter	Description
EXENAME	<u>String expression</u> that specifies the executable name and path. If the executable already exists, it will be overwritten. If the path is not specified, the executable is saved in the active folder. You should provide an EXE extension with the executable name.

### Note

- The .MakeEXE command corresponds to the Make Corel SCRIPT Executable File dialog box in the Corel SCRIPT Editor. Click File, Make EXE.

### Example

The following example creates an executable named CALENDAR.

```
.MakeEXE "C:\Myfiles\Calendar.EXE"
```

You can also specify the line as follows:

```
ReturnValue = .MakeEXE ("C:\Myfiles\Calendar.EXE")
```

---

[{button.AL\(^;cse\\_make;,,,;0,"Defaultoverview",\)} Related Topics](#)

## .SetVisible (Corel SCRIPT Editor)

.SetVisible Show

### Description

Makes the Corel SCRIPT Editor application visible or hidden on your Windows desktop. When the Editor is hidden, it runs in the Windows background and is not visible on screen.

Parameter	Description
Show	Numeric <u>expression</u> that specifies whether Corel SCRIPT Editor is visible or hidden. Set to TRUE (-1) to show the Editor; otherwise set to FALSE (0).

### Notes

- In Windows 95, pressing CTRL+ALT+DEL opens the Close Program dialog box which indicates active applications, both visible and invisible. From Windows NT, pressing CTRL+ALT+DEL opens the Windows NT Security dialog box. Click Task list to see a listing of active applications, both visible and invisible.
- See Executing application commands in the background for more information.

### Example

The following example displays the Corel SCRIPT Editor:

```
.SetVisible -1
```

---

{button.AL(^;cse\_make;;;;;.0."Defaultoverview",)} Related Topics

**String Expression**

A string expression is a combination of literals, string variables, string constants, and string operators that return a string.

**Numeric Expression**

A numeric expression is a combination of numbers, variables, constants, functions, and operators that return a number.

