Corel SCRIPT programming statements and function index

GETTICK function
GETTIMEINFO function
GETTIMER function
GETTOPPOSITION function
GETTYPE function
GETVALUE function
GETVERSION function
GETWIDTH function
GETWINHANDLE function
GLOBAL statement
GOSUB...RETURN statements
GOTO statement
GROUPBOX statement

HELPBUTTON statement
HEX function
HSLIDER statement

IF...THEN...ELSE...ENDIF statements
IMAGE statement
IMAGELISTBOX statement
IMP operator
INCLUDE statement
INPUT statement
INPUT function
INPUTBOX function
INSTR function
INT function

KILL statement

LBOUND function
LCASE function
LEFT function
LEN function
LENGTHCONVERT function
LET statement
LINE INPUT statement
LISTBOX statement
LN function
LOF function
LOG function
Logical Operators
LTRIM function

MESSAGE statement
MESSAGEBOX function
MID function and statement
MKFOLDER statement and function
MOD

MOVE function

NOT operator

OKBUTTON statement
ON ERROR statement
OPEN...APPEND statement
OPEN...INPUT statement
OPEN...OUTPUT statement
OPTIONBUTTON statement
OPTIONGROUP statement
OR operator

PRINT statement
PROGRESS statement
PUSHBUTTON statement

RANDOMIZE function
REDIM statement
REGISTRYQUERY function
Relational Operators
REM statement
REMOVEITEM function
RENAME statement and function
RESET function
RIGHT function
RMFOLDER statement and function
RND function
RTRIM function

SEEK statement
SELECT CASE statements
SETARRAY function
SETBITMAPOFFSET function
SETCURRDATE statement
SETCURRFOLDER statement
SETDOUBLEMODE function
SETEMPTY statement
SETHELPINDEX function
SETHELPPATH function
SETIMAGE function
SETINCREMENT function
SETMAXRANGE function
SETMINRANGE function
SETPRECISION function
SETSELECT function
SETSTYLE function
SETTEXT function
SETTHREESTATE function
SETTICK function

SETTIMER function
SETVALUE function
SETVISIBLE function
SGN function
SIN function
SPACE function
SPINCONTROL statement
SQR function
STARTPROCESS statement and function
STATIC statement
STATUS statement
STEP function
STOP statement
STR function
SUB...END statements

TAN function
TEXT statement
TEXTBOX statement
TOCENTIMETERS function
TOCICEROS function
TODIDOTS function
TOINCHES function
TOPICAS function
TOPOINTS function

UBOUND function
UCASE function
Unary Operators
UNDEF statement

VAL function
VSLIDER statement

WAIT FOR statement
WAIT UNTIL statement
WHILE...WEND statements
WITH...END WITH statements
WITHOBJECT...END WITHOBJECT statements
WRITE statement

XOR operator

Variables

Variables are used to store information in a script. A variable is a value place holder whose name points to an address in the computer's memory where its value is stored. Variable addresses can only hold one value at a time, but during script execution that value can change if you assign another value to the variable, or when you pass the variable to an instruction that can modify its value. Once a script terminates, a variable, and the value it is holding, is lost.

Types of data variables can hold

Variables can hold different types of data. For example, a variable might hold a string or a number. Corel SCRIPT supports nine different variable types: seven numeric types (including a Boolean type and date type), a string type and a variant type. A variable's data type determines the type of data a variable can hold. For more information about data types, see Corel SCRIPT data type summary.

Assigning values to variables

The simplest way to assign a value to a variable is to use the **=** operator as shown in the following example:

```
fox = 1987
car = "Ford"
```

The first statement assigns the numeric value 1987 to the variable **fox**. The second statement assigns the string Ford to **car**.

Declaring variables

A variable must be first declared before you can use it in a script. The above example which assigns values to variables also declares a variable at the same time. This type of variable declaration is called **implicit** and you can find out more about it by seeing Implicitly declaring and assigning values to a variable.

Your second option to declare variables is to use the **DIM**, **GLOBAL** or **STATIC** statements. Using these statements explicitly declare a variable and specify the type of data the variable can hold. For more information, see Explicitly declaring and assigning values to a variable. For information about data types, see Corel SCRIPT data type summary.

{button ,AL(`all_vars;dim;using_constants;Variables_names;;',0,"Defaultoverview",)} Related Topics

Variable names

Variable names can be made up of any combination of letters in the alphabet, both lowercase and uppercase, the numbers 0 through 9, and the underscore character (_). Other rules for variable names follow:

- Variable names are not case sensitive. For example, the variable **ABC** is the same as **abc**, **Abc**, **aBC**, and so on**.**
- The initial character must be a letter, or the underscore character, and can be followed by any combination of letters, numbers, and the underscore character. Variable names cannot include spaces. The following are all acceptable variable names:

  x, X123, Corel_6, TheVariable, a1B2, This_is_a_variable, ThisIsAVariable
- The maximum length of a variable's name is 256 characters.
- A variable cannot have the same name as a Corel SCRIPT statement, function, or operator. See Reserved Words for a complete listing of words that can't be used as a variable name.
- All variable names must be unique in a script procedure. See Variable availability for more information.
- You should use meaningful variable names, and have the names reflect the data that is being stored. For example, if you have a variable that stores a document's page size, you can name the variable **Page_Size**.

Declaring variables

You have two options to create a variable and assign a value to it. The simplest way is to insert a variable name in a script and specify a value for it, as shown in the following example:

```
fox = 1987

car = "Ford"
```

The first statement assigns the numeric value 1987 to the variable **fox**. The second statement assigns the string Ford to **car**. This type of variable declaration is called **implicit** and you can find out more about it by seeing Implicitly declaring and assigning values to a variable.

Your second option to declare variables is to use the **DIM**, **STATIC** or **GLOBAL** statements. Using these statements explicitly declare a variable and is considered better programming practice. For more information, see Explicitly declaring and assigning values to a variable.

{button ,AL(`all_vars;dim;using_constants;;;',0,"Defaultoverview",)} Related Topics

Variable availability

Corel SCRIPT scripts are comprised of three types of procedures:

- main section of a script
- user-defined functions (more than one can exist)
-  user-defined subroutines (more than one can exist)

The availability of a variable is dependent on the procedure the script is executing. The following explains the levels of variable availability in Corel SCRIPT:

- **Global variables** are available anywhere in a running script, but they and their values cease to exist when the script stops running. Global variables are created in the main section of a script and cannot be created within a subroutine or a function. However, they can be used in the execution of any subroutine or function. Use the **GLOBAL** statement to create global variables.
- **Local variables** are available in the procedure in which they are declared. If declared in a subroutine or function, a local variable ceases to exist after the procedure finishes execution and is re-created the next time the subroutine or function is called.
- **Static variables** are declared and assigned values inside a subroutine, or a user-defined function, and are only available while the script executes that subroutine or user-defined function. In contrast to local variables, static variables retain their values after a subroutine, or a function, terminates. The retained value can be used by the script the next time the subroutine or function is called. See **STATIC** for more information.

Note

- You can have variables with the same name in a script, but they cannot exist in the same script procedure (main section, functions, subroutines). For example, you can have a variable called **ABC** in a function and in the main section of a script, but you cannot have two **ABC** variables in the main section of a script.
- Your function and subroutine procedures should be self-contained. A variable required only within a procedure should be either a local or a static variable. Following this custom can make your procedures more modular, enabling you to copy them to other scripts with limited customization. You should avoid using global variables.
- It is a generally accepted programming convention to put variable declaration statements at the beginning of a script's main section, subroutines, or functions.

{button ,AL(`all_vars;static;dim;global;Script_procedures;',0,"Defaultoverview",)} Related Topics

Explicitly declaring and assigning values to a variable

You can use the Corel SCRIPT **DIM**, **GLOBAL** or **STATIC** statements to explicitly declare a variable. By explicitly declaring a variable, you are allocating storage space for it in the computer's memory and setting the type of data it can hold. See Corel SCRIPT data type summary for information about Corel SCRIPT data types.

The following Corel SCRIPT statements declare four variables (**A**, **B**, **C**, and **D**):

```
DIM A%                'declares A as an integer
DIM B$                'declares B as a string
DIM C AS INTEGER   'declares C as an integer
DIM D AS STRING    'declares D as a string
```

Though **C** and **D** are not declared with type-declaration characters, you can still refer to them as **C%** and **D$** in a script. Type declaration characters are optional, but using them is good practice because it makes your scripts easier to read and debug. The variables **A%** and **B$** can also be referred to as **A** and **B**, respectively.

Note

- Formally declaring variables can make your scripts, especially long and complicated scripts, easier to read and modify. It is a generally accepted programming convention to put declaration statements at the beginning of a procedure.
- Once a variable has been created, its type cannot change. However, you can convert a variable to another type by creating another variable. See the following statements for more information: **CBOL**, **CCUR**, **CDAT**, **CDBL**, **CINT**, **CLNG**, **CSNG**, and **CSTR**.
- Explicitly declared variables that aren't assigned a value hold initial values. See Corel SCRIPT data type summary for each data type's initial value.
- During a script run, the availability of a variable to a script changes. See Variable availability for more details.
- You can assign values to variables without using the **LET** statement. The two following lines both assign 5 to **abc%**:
  ```
  LET abc% = 5
  abc% = 5
  ```

{button ,AL(`all_vars;dim;Assigning_values_to_date_variables;;;;',0,"Defaultoverview",)} Related Topics

Implicitly declaring and assigning values to a variable

If you use variables that have not been declared with the Corel SCRIPT **DIM**, **GLOBAL** or **STATIC** statements, you're using implicitly declared variables. You can implicitly declare variables with or without specifying their data type. See Corel SCRIPT data type summary for information about Corel SCRIPT data types.

Implicitly declaring a variable with a specified data type

By assigning a value to an undeclared variable and using a type-declaration character, you can declare a variable's data type. In the following example, the variable **B** is assigned **A**'s value by using the assignment operator (=). The variable **B** is also declared as a long data type by using a type-declaration character.

```
DIM A%   'declares A as an integer
A = 3    'assigns the value of A to 3
B& = A   'declares B as a long with A's value
```

In the following example, **B** is implicitly declared as an integer and holds the value 10, although **A** multiplied by **C** equals 10.8. The variable **B** takes the value 10, because it is implicitly declared as an integer and the precision in the expression **A * C** is lowered after the expression is calculated.

```
DIM A%          'declares A as an integer
DIM C!          'declares C as a single
A = 3           'assigns the value of 3 to A
C = 3.6         'assigns the value of 3.6 to C
B% = A * C      'B is declared as an integer
```

Implicitly declaring a variable without specifying a data type

You can also implicitly declare a variable by assigning a value to it without using a type-declaration character. For example,

```
abc = -400444.4                 'assigns a number
def = "This is a string"    'assigns a string
```

Variables declared in this manner are set to the Corel SCRIPT variant data type. Since variants can hold any other Corel SCRIPT data type as a subtype, the variable **abc** is a variant of subtype double. The variable **def** is a variant of subtype string. For more information, see Variants.

If you assign a numeric value to an undeclared variable using previously declared variables or constants, Corel SCRIPT implicitly declares the variable as a variant. The variant takes on subtype of the data type of the expression on the right side of the equals sign. The following example illustrates this situation:

```
DIM A%    'declares A as an integer
DIM C!    'declares C as a single
A = 3     'assigns the value of 3 to A
C = 3.6   'assigns the value of 3.6 to C
E = A * C
```

In the above example, **A** is declared as an integer and is set to 3. The variable **C** is declared as a single and is set to 3.6. The last line implicitly declares the variable **E** and assigns it the product of **A** and **C**. Since the value with the highest precision on the right side of the equal sign is single, **E** is declared a variant of the type single. To see the summary of the levels of precision in Corel SCRIPT, see Corel SCRIPT data type summary.

Note

- Any implicitly declared variable is local to the procedure it was declared in. See Variable availability for more details.
- There is no need to explicitly declare a variable as a variant, unless you want to format your script for better readability.
- Once a variable has been created, its type cannot change. However, you can convert a variable to another type by creating another variable. See the following statements for more information: **CBOL**, **CCUR**, **CDAT**, **CDBL**, **CINT**, **CLNG**, **CSNG**, and **CSTR**.
- Using variants instead of explicitly declared data types can slow down the execution of a program because of all the data types, variants use the most memory. See Corel SCRIPT data type summary for more information.
- Explicitly declaring variables can make your scripts, especially long and complicated scripts, easier to read and modify.

{button ,AL(`all_vars;dim;;;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT data type summary

A variable's data type determines the type of data a variable can hold. Corel SCRIPT supports the data types listed in the following table. Each data type has a type declaration character (where applicable), storage space, range, and initial value noted.

Data Type	Character	Byte Storage	Range	Initial Value
String	**$**	4 + 1 per character	0 to 4 billion characters (approximate)	""
Boolean		1	TRUE (-1) or FALSE (0)	FALSE
Integer	**%**	2	-32,768 to 32,767 (whole numbers)	0
Long	**&**	4	-2,147,483,648 to 2,147,483,647 (whole numbers)	0
Single	**!**	4	Negative Values: -3.402823E38 to -1.401298E-45 Positive Values: 1.401298E-45 to 3.402823E38	0
Double	#	8	Negative Values: -1.79763913486232E308 to -4.94065645841247E-324 Positive Values: 4.94065645841247E-324 to 1.79763913486232E308	0
Date		8	1 to 2958465 (as a serial number)	1 (as a serial number)
			31/12/1899 00:00:00.0000 to 31/12/9999 23:59:59.9999 (as a date value)	31/12/1899 00:00:00.0000 (as a date)
			(dd/mm/yyyy hh:mm:ss.ssss)	
Currency	@	8	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	0.0000
Variant (numeric type)		10	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	EMPTY (0)
Variant (string type)		10 + 1 per character	0 to 4 billion characters (approximate)	EMPTY ("")

Note

- The data types are listed in order of precision from the lowest to the highest.
- Boolean, Date, and Variant data types don't use type declaration characters, but can still be explicitly declared using the **DIM** statement:

```
DIM x AS BOOLEAN
DIM y AS DATE
DIM z AS VARIANT
```

- Instead of the **DIM** statement, you can also use the **GLOBAL** or **STATIC** statements to declare a variable.

{button ,AL(`all_vars;dim;vars_convert;;;;',0,"Defaultoverview",)} Related Topics

Boolean data type

Boolean data types are explicitly declared using the **BOOLEAN** keyword and do not use a type declaration character. Booleans, which are numeric variables, can only equal TRUE (-1) or FALSE (0). If you set a Boolean variable to anything but -1 or 0, the Boolean variable interprets the value as TRUE and resets to -1. Booleans are used to test conditional expressions such as those used with IF...THEN...ELSE...ENDIF. Of the eight numeric variables in Corel SCRIPT, Booleans are of the lowest precision.

Integer data type

Integer data types are explicitly declared using the **%** type declaration character or the **INTEGER** keyword. Integers, which are numeric variables, range in value from -32,768 to 32,767 and are whole numbers only. Of the eight numeric variables in Corel SCRIPT, integers are of the second-lowest precision.

Long data type

Long data types are explicitly declared using the **&** type declaration character or the **LONG** keyword. Longs, which are numeric variables, range in value from -2,147,483,648 to 2,147,483,647, and are whole numbers only. Of the eight numeric variables in Corel SCRIPT, longs are of the third-lowest precision.

Single data type

Single data types are explicitly declared using the **!** type declaration character or the **SINGLE** keyword. Singles, which are numeric variables, range in value from -3.402823E38 to -1.401298E-45 for negative numbers and 1.401298E-45 to 3.402823E38 for positive numbers. Of the eight numeric variables in Corel SCRIPT, singles are of the fourth-highest precision.

Double data type

Double data types are explicitly declared using the # type declaration character, or the **DOUBLE** keyword. Doubles, which are numeric variables, range in value from -1.79763913486232E308 to -4.94065645841247E-324 for negative numbers and 4.94065645841247E-324 to 1.79763913486232E308 for positive numbers. Of the eight numeric variables in Corel SCRIPT, doubles are of the third-highest precision.

When a double variable type is used with a relational operator, it is temporarily recast as a single variable type.

Currency data type

Currency data types are explicitly declared using the @ type declaration character or the **CURRENCY** keyword. Currency variables range in value from -922,337,203,685,477.5808 to 922,337,203,685,477.5807. Of the eight numeric variables in Corel SCRIPT, currency variables are of the second-highest precision. In cases where exactness is important, such as calculations involving finance, money, and fixed-points, you should use currency variables.

Date data type

Date data types are explicitly declared using the **DATE** keyword and do not use a type declaration character. Date variables hold date and time values that range from 1 to 2958465 (as a serial number) or 31/12/1899 00:00:00.0000 to 31/12/9999 23:59:59.9999 (as a date value). You can use dates outside this range, but they are not supported by Corel SCRIPT and may lead to errors. A serial value of 1 is equal to 1 day or a 24-hour period.

See Using date and time for more information about Corel SCRIPT and dates.

String data type

String data types are explicitly declared using the **$** type declaration character or the **STRING** keyword. Strings can hold 0 to 32,765 characters. Characters include letters, numbers, punctuation, and spaces. Strings can use any of the 256 ANSI characters for Windows applications (code 0 to 255).

See the Corel SCRIPT Character Map for more information.

Variant data type

Implicitly declaring a variable sets its type to Variant. You can also explicitly declare a variant using the **VARIANT** keyword. Variants can contain any other type of Corel SCRIPT data, but do so in a less efficient manner than actually declaring a specific data type. Since variants use more storage space than other data types, it is advisable to explicitly declare a variables type. This can help speed up long or recursive scripts. See Variants for more information.

Corel SCRIPT type declarations

Data Type Name	Character
INTEGER	**%**
LONG	**&**
SINGLE	**!**
DOUBLE	**#**
CURRENCY	@
STRING	**$**
BOOLEAN	**None**
DATE	**None**
VARIANT	**None**

Corel SCRIPT naming convention

The following rules should be kept in mind when naming variables, constants, functions, subroutines, parameters, and arrays:

- Names can be made up from any combination of letters in the alphabet (both lowercase and uppercase), the numbers 0 through 9, and the underscore character (_). The initial character must be a letter or the underscore character, and can be followed by any combination of letters, numbers, and the underscore character. The following are all valid names:
  - **x, X123, Corel_6, TheFunction, a1B2, This_is_a_SUB, ThisIsAConstant, My_2_ARRAY**
- Names are not case sensitive. For example, the name **ABC** is the same as **abc**, **Abc**, **aBC**, and so on**.**
- The maximum length of a name is 256 characters.
- All names must be unique in a script file. See Reserved Words for a complete listing of words that can't be used as a name.
- Names cannot include spaces.

Variants

Corel SCRIPT variables take on the variant data type if they're implicitly declared; that is, not declared using the **DIM**, **GLOBAL** or **STATIC** statements, or a type-declaration character. The variant data type can contain, or hold, all other Corel SCRIPT data types. The Corel SCRIPT data types, when used with variants, are called data subtypes. For information on the other data types in Corel SCRIPT, see the Corel SCRIPT data type summary.

The subtype that a variant uses depends on the precision of the expression used to set the variable's value. The highest precision data type used in the expression assigning a value to a variant becomes the variant's data subtype. For example, in the following script statements, **X** is declared as an integer and **Y** is declared as a long. The last line creates a variant variable **Z** of subtype long, since the highest precision in the numeric expression of **X + Y** is long.

```
DIM X%
DIM Y&
X = 2
Y = 40000
Z = X + Y
```

A variant's subtype does not change unless an expression using a higher precision is used to set the variant's numeric value. If the above example was continued, and the following statements are executed, the variant variable's subtype would be reset to a single.

```
W! = 1.1 ' w is set as a Single
Z = Z + W
```

The first line creates the variable **W** of single type. The second line adds **W** to **Z**, changing its precision and subtype from long to single.

If you assign a whole number to an undeclared variable, or a fractional number to an undeclared variable, the resulting variant is set to long type or double type, respectively. For example,

```
A = 3       'sets A as variant of type long
B = 3.3     'sets B as variant of type double
```

For more information about how variants are used in numeric expressions, see Precision in expressions using mixed data types.

Empty variant types

In addition to the variants holding all other Corel SCRIPT data types, they can also hold the **Empty** subtype. A variant uses the Empty subtype if the variant is explicitly declared using the VARIANT data type name, or by using the **SETEMPTY** statement. An Empty variant contains no valid data in both a numeric and string context.

Note

- You can determine a variant's subtype by using the **GETTYPE** function.
- Using variants instead of explicitly declared data types can slow down the execution of a program because of all the data types, variants use the most memory. See Corel SCRIPT data type summary for more information.
- Since Corel SCRIPT, by default, sets implicitly declared variables to the variant type, there is no need to explicitly declare a variable as a variant, unless you want to format your script for better readability.
- Once a variable has been created, its type cannot change. However, you can convert a variable to another type by creating another variable. See the following statements for more information: **CBOL**, **CCUR**, **CDAT**, **CDBL**, **CINT**, **CLNG**, **CSNG**, and **CSTR**.
- Any implicitly declared variable is local to the procedure it was declared in. See Variable availability for more details.
- Explicitly declaring variables can make your scripts, especially long and complicated scripts, easier to read and modify.

{button ,AL(`setempty;gettype;all_vars;dim;gettype;precision_expressions;;;',0,"Defaultoverview",)} Related Topics

Strings

In Corel SCRIPT, a string is the only data type that is not numeric; that is, a string is a series of characters that is treated as a unit of information. Strings are explicitly declared using the **$** suffix or the data type name **STRING**. Strings must be enclosed in double quotation marks, and can hold up to 32,765 characters including lower and uppercase letters, numbers, and punctuation marks. Strings can use any of the 256 ANSI characters for Windows applications (code 0 to 255).

The following example declares the string variable **stringVar$**:

```
stringVar$ = "This is a sample string."
```

Special Characters

Punctuation and certain character codes for items such as tabs and hard returns cannot be directly entered within a string's double quotation marks. The **CHR** function is often used to add these special characters. For example, to add double quotation marks to a string variable, use character 34:

```
s$ = CHR(34) + "This will be in double quotes." + CHR(34)
MESSAGE s$
```

You can also use this function to add a return and a line feed within a string, with character 13 and 10, respectively:

```
s$ = "String 1" + CHR(13) + "String 2"
MESSAGE s$
```

The result will be the two strings on separate lines, as displayed in the message box.

See the Corel SCRIPT character map or **CHR** for more information about using special characters in strings.

{button ,AL(`cs_strings_fns;using_variables;corel_script_data_type_summary;;;',0,"Defaultoverview",)} Related Topics

Constants

A constant is a data item that does not change for the duration of a script run. Using a constant instead of a variable ensures that you don't accidentally change a value.

The two following script statements each create a constant (NATURAL_LOG and PI):

CONST NATURAL_LOG# = 2.71828182845 'creates a constant for the base of the natural logarithm

GLOBAL CONST PI# = 3.14 'creates a global constant for pi

By giving your declared constants meaningful names, you can also make your scripts easier to read and debug.

Constant availability

Corel SCRIPT scripts are comprised of three types of procedures:

- main section of a script
- user-defined functions (more than one can exist)
- user-defined subroutines (more than one can exist)

The availability of a constant is dependent on the procedure the script is executing. The following explains the levels of constant availability in Corel SCRIPT:

- **Global constants** are available anywhere in a running script, but they and their values cease to exist when the script stops running. Global constants are created in the main section of a script and cannot be created within a subroutine or a function. However, they can be used in the execution of any subroutine or function.
- **Local constants** are available in the procedure in which they are declared. If declared in a subroutine or a function, a local constant ceases to exist after the procedure finishes execution and is re-created the next time the subroutine or function is called.

Note

- You can have constants with the same name in a script, but they cannot exist in the same script procedure. For example, you can have a constant called ABC in a function and in the main section of a script, but you cannot have two ABC constants in the main section of a script.
- Your function and subroutines procedures should be self-contained. A constant required only within a procedure should be a local constant. Following this advice can make your procedures more modular, enabling you to copy them to other scripts with limited customization.
- It is a generally accepted programming convention to put constant declaration statements at the beginning of a script's main section, subroutines, or functions.

{button ,AL(`using_variables;const;DEFINE;;;;',0,"Defaultoverview",)} Related Topics

Operators

An operator is a symbol or word that performs a function on one or more expressions. Operators compare expressions, link words together, and perform mathematical functions.

Corel SCRIPT supports the following operators:

- Arithmetic
- Assignment
- Bitwise
- Concatenation
- Logical
- Relational
- Unary

{button ,AL(`all_operators;;;;;',0,"Defaultoverview",)} **Related Topics**

Operator precedence in Corel SCRIPT

In Corel SCRIPT, expressions with more than one operator are evaluated by a predetermined order of execution, known as operator precedence. Operations with a higher precedence are followed by operations with a lower precedence. Operators with the same precedence are evaluated left to right.

Operations within parentheses are evaluated before operations outside of parentheses. Operators and expressions nested with parentheses are evaluated from the innermost to the outermost.

The following table lists the order of precedence from the highest to the lowest:

The following table shows operator precedence, which is applied to expressions with two or more operators.

Order	Operator
1	parentheses ( )
2	unary +, unary - , NOT
3	exponentiation (^)
4	multiplication (*), division (/)
5	integer division (\\)
6	modulus (MOD)
7	addition (+), subtraction (-)
8	shift left (<<), shift right (>>)
9	concatenation + , &
10	relational operators (=, <>, >, <, =>, >=, <=, =<)
11	AND
12	OR
13	XOR
14	EQV
15	IMP

{button ,AL(`all_operators;;;;;',0,"Defaultoverview",)} **Related Topics**

Expressions

Variables and constants are joined by operators to form expressions. Expressions often compute values that determine how control statements direct script execution. There are two main types of expressions in Corel SCRIPT: numeric and string.

Numeric expression

A numeric expression is a combination of numeric constants or variables, or a combination of the two joined by a operator. A numeric operator is a word or symbol that operates on numeric data. Given that x equals 3, the following examples are valid numeric expressions.

5
Numeric constant.

X
Variable containing a numeric value.

x * 5
Expression that uses an arithmetic operator to multiply x and 5.

+5
Unary plus constant.

-(x + 10)
Unary minus expression (negate the result of x plus 10).

String expression

A string expression is a combination of the two concatenated by the plus operator (+), or a combination of the two separated by the minus operator (-), or compared by a relational operator such as > (greater than). String constants and strings variables must be enclosed in double quotation marks. The following examples are valid string expressions (notice the space after Joe and before Jr.).

"John Doe"
String constant

z = "Joe " & " Doe"
String expression assigned to variable z (z equals "Joe Doe").

x = z & ", Jr."
String expression assigned to variable x (x equals "Joe Doe, Jr").

Note

- You can use scientific notation to express numbers. For example, 123.45E3 equals 123,450.

{button ,AL(`using_strings;variants;Using_Variables;Using_constants;Using_operators;Using_arrays;',0,"Defaultoverview",)} Related Topics

Numeric expression

- In Corel SCRIPT, a numeric expression is a combination of numbers, variables, constants, functions, and operators that return a number.

String expression

- In Corel SCRIPT, a string expression is a combination of literals, string variables, string constants, and string operators that return a string.

Precision in expressions using mixed data types

A variable's data type determines the type of data a variable can hold. Each data type has a level of precision associated with it. The following is a list of all the Corel SCRIPT data types in order from the lowest to the highest level of precision:

String

Boolean

Integer

Long

Single

Double

Date

Currency

Variant

In operations of mixed data types, all data types are temporarily converted to the data type with the highest level of precision. For example,

```
DIM A%     'declares A as an integer
DIM C!     'declares C as a single
A = 3      'assigns the value of 3 to A
C = 3.6    'assigns the value of 3.6 to C
E = A * C
```

In the above example, **A** is declared an integer and is set to 3. The variable **C** is declared a single and is set to 3.6. The last line implicitly declares the variable **E** and assigns it the product of **A** and **C**. Since the value with the highest level of precision on the right side of the equal sign is a single, **E** is declared a variant of the subtype single.

The following are exceptions to the rule of highest precision:

Concatenation operator (&)

Since the concatenation operator & is a string-only operator, data is converted to strings before the concatenation occurs. For example, if you concatenate a string and a number, the number is converted to a string.

```
x  =  "A" & 1
x equals the string "A1"
```

Division operator (/)

The division operator is a fractional operator meaning that all non-fractional data type operands are temporarily converted to the double data type.

```
DIM A%     'declares A as an integer
DIM C&     'declares C as a long
A = 3      'assigns the value of 3 to A
C = 12     'assigns the value of 12 to C
E = C / A
```

In the above example, **A** is declared as an integer and is set to 3. The variable **C** is declared as a long data and is set to 12. The last line implicitly declares the variable **E** and assigns it the result of **C** divided by **A.** Since both **A** and **C** are temporarily converted to the double data type, **E** is declared a variant of the subtype double.

Non-fractional operators

Integer division (\), right shift (>>), and left shift (<<) are non-fractional operators. All non-whole number operands that use these operators are temporarily converted to a long data type before the operation is executed.

Relational operators

Relational operators (>, <, >=, <=, <> and =) are used to compare two operands (numeric or string expressions). However, relational operators can only return a Boolean result TRUE (-1) or FALSE (0). Click for more information about relational operators.

Variants

Certain operations may promote a variant's subtype to a higher type. Corel SCRIPT attempts to promote a subtype to prevent an overflow error from occurring. Overflow errors occur when a variable is assigned a value outside of its data type range. For example, if a variant of subtype integer is assigned the value 33,333, Corel SCRIPT attempts to promote the subtype to a long. The following table lists the data subtypes Corel SCRIPT can promote.

Original subtype	Promoted subtype level
String	Overflow error (no subtype promotion)
Boolean	Overflow error (no subtype promotion)
Integer	Long
Long	Double
Single	Double
Double	Overflow error (no subtype promotion)
Date	Overflow error (no subtype promotion)

Currency	Overflow error (no subtype promotion)

See Corel SCRIPT data type summary for more information about a data type's valid value range.

Note

- In expressions, a whole number is treated as a Long while a fractional number is treated as a Double. To treat a whole number as an Integer, use the data type character **%**. For example, 123%. To treat a fractional number as a Single, use the data type character **!**. For example, 123.45!.
    - You can also treat a fractional number as Currency by using the data type character @. For example, 123.45@.

{button ,AL(`variants;Variable_availability;Explicitly_declaring;Implicitly_declaring;cs_expressions;;',0,"Defaultoverview",)} Related Topics

Arrays

An array is a variable type containing a group of values of the same data type in an ordered list format. For example, the following Corel SCRIPT statements declare and assign values to the string array color:

```
DIM color$(5)
color$(1) = "black"
color$(2) = "red"
color$(3) = "white"
color$(4) = "blue"
color$(5) = "green"
```

To use an array, you must address a specific element of the array. For example, to use the fourth string in the array mentioned above, you would use the variable **color$(4)**.

Arrays are useful in defining control values in Corel SCRIPT dialog list boxes and with the **FOR...NEXT** statements.

In the following example, the **FOR...NEXT** loop is used to assign the numbers 1 through 50 to the corresponding elements of an integer array:

```
DIM numberArray%(50)
FOR i% = 1 TO 50
    numberArray%(i%) = i%
NEXT i%
```

Note

- Arrays are created with the **DIM** (for "dimension") statement.
- Arrays can only hold one data type. Click  for more information about data types.
- See Multi-dimensional arrays to create arrays of more than one dimension.
- You can change the number of elements in an array by using the **REDIM** statement.
- You can only assign values to existing array elements. The following example will cause an error:

```
DIM A$(10)          'creates a string array of 10 elements
A$(11)="string value"          'attempts to assign a value to A's eleventh array element
```

{button ,AL(`REDIM;listbox_example;FOR_NEXT;DIM;LBOUND;UBOUND;multi_dimensional_arrays;',0,"Defaultoverview",)} Related Topics

Example

Multi-dimensional arrays

For multi-dimensional arrays
DIM array_name{%|&|!|#|@|$} (l_bound TO u_bound, l_bound TO u_bound, ...)
DIM array_name(l_bound TO u_bound, l_bound TO u_bound, ...) AS type

Corel SCRIPT also features multi-dimensional arrays to create tables with more than one column.

Syntax	Definition
array_name{%|&|!|#|@|$}	Specifies the name of the array and follows the Corel SCRIPT naming convention. A type-declaration character must follow the name in the case of an array.
array_name	Specifies the name of the array and follows the Corel SCRIPT naming convention.
u_bound	The upper bound of the array, expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **u_bound**) is used.
l_bound	The lower bound of the array, expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **u_bound**) is used.
type	Declares the variable's or array's type with a type declaration name.

{button ,AL(`using_arrays;dim;lbound;ubound;;',0,"Defaultoverview",)} Related Topics

Example for multi-dimensional arrays

The three following examples create 6-by-6 arrays containing 36 elements:

DIM array1$(6, 6)

DIM array2!(-2 TO 3, 2 TO 7)

DIM arr%(0 TO 5, 0 TO 5)

The last example shows how you can use multi-dimensional arrays. If you were to display **arr%** as a table, it would look like this:

1, 1	1, 2	1, 3	1, 4	1, 5	1, 6
2, 1	2, 2	2, 3	2, 4	2, 5	2, 6
3, 1	3, 2	3, 3	3, 4	3, 5	3, 6
4, 1	4, 2	4, 3	4, 4	4, 5	4, 6
5, 1	5, 2	5, 3	5, 4	5, 5	5, 6
6, 1	6, 2	6, 3	6, 4	6, 5	6, 6

The cells represent array elements and the numbers represent the array index numbers or subscripts. For example, the cell (5, 3) represents **arr%(5, 3)** which represents two integer variables.

The following example creates Pascal's Triangle in a message dialog box using a multi-dimensional array and the FOR...NEXT loop. (Pascal's Triangle is a triangle array of integers in which each number is the sum of the numbers above it in the preceding row. The apex of the triangle is 1.)

```
DECLARE FUNCTION factorial%(a%) 'create an integer formula
DIM arr%(0 TO 5, 0 TO 5)   'this is a 6 by 6 array

FOR i% = 0 TO 5
    FOR j% = 0 TO 5
        arr(i, j) = factorial(i) / (factorial(j)*factorial(i-j))
    NEXT j
NEXT i
'
FOR i% = 0 TO 5
    FOR j% = 0 TO 5
        IF j <= i THEN mess$ = mess$ + CHR(9) + CSTR(arr(i, j))
    NEXT j
    mess$ = mess$ + CHR(13)
NEXT i
MESSAGE mess$
'
FUNCTION factorial%(a%)
    temp% = 1
    IF a > 0 THEN
        FOR lop% = 1 TO a
            temp% = temp% * lop
        NEXT lop
    END IF
    factorial = temp
END FUNCTION
```

{button ,AL(`example_dim;dim;ubound;lbound;;using_variables;using_arrays;',0,"Defaultoverview",)} **Related Topics**

Script procedures

A simple Corel SCRIPT script executes in a linear manner: each statement is executed on a line-by-line basis up to the last script statement. If you have a group of instructions that will be repeated in a script, create a user-defined procedure (function or subroutine) for those instructions. The instructions are written once in the script, and can be called from different places within the script. If the instructions are changed, the changes take effect everywhere. Using these user-defined procedures can make your scripts easier to change and debug.

Corel SCRIPT scripts are comprised of three types of procedures:

- main section of a script (one per script)
- user-defined functions (optional and more than one can exist)
- user-defined subroutines (optional and more than one can exist)

Functions and subroutines are groups of Corel SCRIPT statements that are executed when the procedure is called by another Corel SCRIPT statement. Both types of procedures are useful in cases where a group of instructions will be repeated. The instructions are written once in the script, and can be called from different places within the script or with different parameters. You can have more than one of each type of procedure in a script.

Although, user-defined subroutines and functions are both Corel SCRIPT procedures that execute instructions, functions can also be used to return values to a script that can be either assigned to a variable or compared with other expressions.

Note

- Your function and subroutines procedures should be self-contained; a variable is only required within a procedure should be a local or static variable. This will make your procedures more modular, enabling you to copy them to other scripts with limited customization.
- You can also use the **GOSUB** statement to create a simple subroutine.

{button ,AL(`function_end_function;sub_end_sub;goto;gosub_return;const;global;call;Variable_availability;Using_functions_subroutines;Executing_script_files;declare;',0,"Defaultoverview",)} Related Topics

User-defined functions

User-defined functions can have zero or more parameters which receive values from a function calling statement. Functions can also be used to return a value. Function names follow the Corel SCRIPT naming convention. The following rules apply to functions:

- Functions begin with the word **FUNCTION** and end with **END FUNCTION.**
- Functions do not execute unless called.
- Calling statements consist of a function name, which can be followed by zero or more parameter values that pass to the function. If there are no parameters, empty parentheses must follow the function name. Optionally, you can use the **CALL** to call a user-defined function.
- The number of parameters in a calling statement must match the number of function parameters.
- Functions can be defined anywhere in a script except inside another procedure.
- Functions can be called from a procedure, or a function can call itself (functions are recursive).
- Global variables can be used in any script user-defined procedure. Local variables are only available in the user-defined subroutine they are declared in. See Variable availability for more information about using variables in user-defined subroutines.
- Functions can include line labels, or line numbers which are not available outside the subroutine. For more information about line labels and line numbers, see the **GOTO** statement.

{button ,AL(`Passing_parameters;;;;;',0,"Defaultoverview",)} Related Topics

User-defined subroutines

User-defined subroutines can have zero or more parameters, which receive a value from a subroutine calling statement. A subroutine cannot return a value. Subroutine names follow the Corel SCRIPT naming convention. The following rules apply to subroutines:

- Subroutines begin with the word **SUB** and end with END SUB.
- Subroutines cannot execute unless called.
- Calling statements consist of a subroutine name, which can be followed by zero or more parameter values that are passed to the subroutine. Optionally, you can use the **CALL** statement to call a user-defined subroutine.
- The number of parameters in a calling statement must match the number of subroutine parameters.
- A subroutine can be defined anywhere in a script, except inside another procedure.
- Subroutines can be called from a procedure, or a subroutine can call itself (recursive).
- Subroutines can include line labels, or line numbers which are not available outside the subroutine. For more information about line labels and line numbers, see the **GOTO** statement.
- Global variables can be used in any script user-defined procedure. Local variables are only available in the user-defined subroutine they are declared in. See Variable availability for more information about using variables in user-defined subroutines.
- You can also use the **GOSUB** statement to create a simple subroutine that doesn't use parameters.

{button ,AL(`Passing_parameters;;;;;',0,"Defaultoverview",)} Related Topics

Flow control statements

Corel SCRIPT scripts generally execute in a linear manner; that is, in the order in which the script instructions are listed. Flow control statements allow you to dictate how a script can be executed. Using flow controls can make your scripts more flexible and efficient. Two of the most important types of flow control statements in Corel SCRIPT are conditional statements and looping statements.

Conditional statements

Conditional statements execute other script statements, or a block of statements, when an expression meets a condition of TRUE or FALSE, or when a variable matches a constant. Conditional statements are useful when you want to provide a user with a list of options. Corel SCRIPT includes the following conditional statement constructs:

IF...THEN...ELSE...ENDIF

ON ERROR

SELECT CASE

Looping statements

Corel SCRIPT looping statements execute script instructions or blocks of script instructions a specified number of times, until an expression is TRUE, or while an expression is TRUE. Corel SCRIPT supports the following standard BASIC looping constructs:

DO...LOOP

FOR...NEXT

WHILE...WEND

Other flow control statements

You can use flow control statements to

- Repeat statements while, or until, an expression meets a condition of TRUE or FALSE.
- Repeat statements a specific number of times.
- Go to a different line in the script.
- Terminate a script's execution.
- Direct script execution to a Corel application.

{button ,AL(`cs_flows;using_operators;using_variables;;;',0,"Defaultoverview",)} Related Topics

Comments

You can use comments in a script to describe how the script works. If you plan on sharing your scripts with other users, include comments in the script to explain your techniques. Comments also help if you have to modify a script weeks after it is written.

To include a comment in a script, use the **REM** (for "remark") statement. The comment must stand alone on the line, and the REM must be at the beginning of the comment (spaces and tabs are ignored at the beginning of a line).

To include a comment on a line that also contains a script instruction, type an apostrophe (') after the instruction and type the comment after the apostrophe. The script compiler will ignore any characters to the right of an apostrophe, until it reaches the end of the line. The following example shows comments on every script line:

```
REM     The next 2 lines contains a comment
BEEP    'Sounds a computer beep
' This is the same as REM statement
```

To have a script description appear in your Corel application's status bar when a menu item or toolbar button assigned to the script is highlighted, make sure that the first line of the script contains a comment. For example,

```
REM Customization script
```

If the first line of the script is not the description, a default description will be used. Keep the description brief so that it fits on the status bar. When you save a recording, Corel SCRIPT adds two description lines at the top of the script for you. (Not all Corel applications support recording commands.) If a script's first line, second line, or both are REM statements, the comments are also displayed in a Corel application's Run Script dialog box if the script is specified.

{button ,AL(`REM;Formatting_a_script;;;;',0,"Defaultoverview",)} **Related Topics**

Using date and time

Corel SCRIPT provides statements, commands, and functions to manipulate dates and time. Dates are one of Corel SCRIPT's variable types. They are explicitly declared using the **DATE** keyword. Date variables hold date and time values that range from 1 to 2958465 (as a serial number) or 31/12/1899 00:00:00.0000 to 31/12/9999 23:59:59.9999 (as a date value). You can use dates outside this range, but they are not supported by Corel SCRIPT and can lead to errors in script execution.

Serial numbers represent both date and time values. Numbers to the left of the decimal represent days, and numbers to the right of the decimal represent time. If you omit decimal numbers, time is set to 12:00:00 A.M.

The following table shows examples of serial numbers using whole and decimal numbers to represent dates and time:

Serial number	Date and time represented
1	December 31, 1899, 12:00:00 A.M.
2	January 1, 1900, 12:00:00 A.M.
2.25	January 1, 1900, 6:00:00 A.M.
16229.2292	June 6, 1944, 5:30:00 A.M.
23337.4375	November 22, 1963, 10:30 A.M.
25882.6701	November 10, 1970, 4:04:57 P.M.
35064.9999	December 31, 1995 11:59:59 P.M.
35065	January 1, 1996, 12:00:00 A.M.
35065.5	January 1, 1996, 12:00:00 P.M.

The following table shows the decimal value for the hours of the day:

Decimal serial number	Hour of the day represented
.0	12 A.M.
.125	3 A.M.
.25	6 A.M.
.375	9 A.M.
.5	12 P.M.
.625	3 P.M.
.75	6 P.M.
0.875	9 P.M.

Each hour is approximately equal to .041666

Note

- Formatting used to display dates and time is set in the Windows Control Panel. In Windows 95, see Regional settings for formatting information; in Windows NT, see International settings.

{button ,AL(`cs_date_time;Corel_SCRIPT_data_type_summary;;;;',0,"Defaultoverview",)} **Related Topics**

Assigning values to date variables

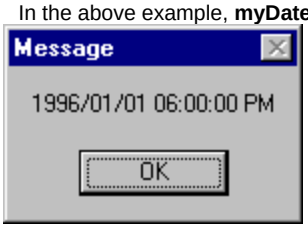
You have two options to assign a date and a time to a date variable. One option is to use the serial number. For example:

DIM myDateVar AS DATE

myDateVar = 35065.75

MESSAGE myDateVar

In the above example, **myDateVar** is set to January 1, 1996, 6:00 P.M., as shown in the following message box:



The second option is to use a string to assign a date and a time to a date variable. The following examples all set **myDateVar** to January 1, 1996, 6:00 P.M.: gargoyle

myDateVar = "01/01/96 18:00"

myDateVar = "January 1 1996 6:00 PM"

myDateVar = "1 January 96 6:00 PM"

myDateVar = "01-01-96 18:00"

The Corel SCRIPT date variable uses predefined logic to determine how to convert strings into dates. In ambiguous cases, for example, when a portion of a string could be a day, a month, or a year, Corel SCRIPT refers to the system's date settings to determine how to convert the string.

Note

- Formatting used to display dates and time is set in the Windows Control Panel. In Windows 95, see Regional settings for formatting information; in Windows NT, see International settings.

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} **Related Topics**

Toolbars dialog box

You have complete control over your toolbars. With simple mouse actions, you can dock, undock, resize, and move your toolbars anywhere on the screen.

The Toolbars dialog box enables you to controls which toolbars are displayed on the screen at any given moment.

{button ,AL(`cs_custom_ov_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

Displays the available toolbars. Enable the checkbox next to a toolbar to activate it. Click the toolbar's name tag to rename it.

Creates a new toolbar. Click Customize to add buttons to the new toolbar.

Deletes a custom toolbar, or resets a built-in toolbar.

Opens the Customize dialog box, where you can change the configuration of your toolbar buttons.

Click this button to change the size of the buttons on the toolbar and whether the popup window describing the function of the button displays when you rest the mouse pointer over it.

Drag the bar to change the size of the button.

Drag the bar to change the size of border around the toolbar.

Enable this option to display titles on floating toolbars.

Displays the system information for the chosen category.

Opens the System Info dialog box where you can get information about your system, display, printing, Corel EXEs & DLLs and system DLLs.

Displays the disk space available on the drive where the program is installed.

Displays the name of the registered user and the serial number.

Displays copyright information.

Displays the version of the product currently installed.

Displays the name of the product.

Double-click to open the credits window. To exit the credits and return to your program, click the ESC button.

Provides a list of system information categories. Click one of the following:

System	Displays information about your computer For example, Windows version or processor.
Display	Displays information about your monitor. For example, driver,or driver version.
Printing	Displays information about installed printers.
Corel EXEs and DLLs	Displays information all of the Corel EXEs and DLLs.
System DLLs:	Displays all of the system DLLs.

Saves all system information as SYSINFO.TXT. Once it's saved, a message box appears informing you of the location of the saved file.Double-click to open the credits window. To exit the credits and return to your program, click the ESC button.

Double-click to open the credits window. To exit the credits and return to your program, click the ESC button.

Displays information about the contents of the file.

Using the Corel SCRIPT online Help

If you want to learn how to record and playback your actions with a Corel application, see the Reference topic in your application's online Help. You can also learn how to save your recordings as scripts (sometimes called macros) and find out how to assign scripts to keystrokes, menus, and toolbar buttons.

This online Help can provide all the information you need if you're ready to go beyond the features noted above. This information is intended for both novice and experienced script or macro writers and programmers. The information is broken into the following categories:

Corel SCRIPT Basics

This section provides an overview of what Corel SCRIPT is and how you can use it. It also provides information regarding the syntax and documentation conventions used in Corel SCRIPT.

Corel SCRIPT Concepts

This section introduces Corel SCRIPT programming language concepts. You should view this section if you are new to script writing. If you're an script writer or programmer, you may want to skip to the next section.

Corel SCRIPT application commands and functions & Corel SCRIPT programming statements and functions

These two sections explain the syntax and purpose of all Corel SCRIPT application commands and programming statements.

Corel SCRIPT Editor

Explains the features of the Corel SCRIPT Editor and how it can be used to quickly create and edit your scripts. This section also explains about creating and editing custom dialog boxes.

Custom dialog boxes

This section explains how to use custom dialog boxes in your scripts.

How to

Provides procedural information for using the Corel SCRIPT Editor and creating and editing custom dialog boxes.

Advanced Corel SCRIPT features

Describes the advanced features available in Corel SCRIPT to develop and use DLLs and executables. This section is aimed at experienced Windows programmers and third-party developers.

Reference

Provides reference information such as error codes, warning messages, character map and a glossary.

Note

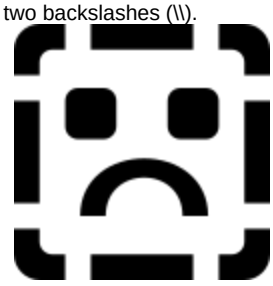
- Corel DRAW 6 doesn't support recording commands and actions but you can write scripts for it.

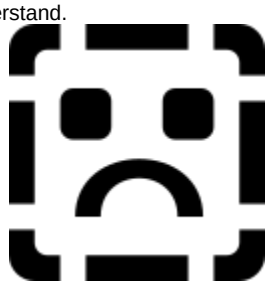
{button ,AL(`Documentation_syntax_conventions;online_program_cs;online_application_cs;;;',0,"Defaultoverview",)}
Related Topics

Changes in Corel SCRIPT from version 6 to 7

This topic outlines the changes to Corel SCRIPT since the release version 6 which was included with CorelDRAW 6, Corel PHOTO-PAINT 6, CorelFLOW 3, and CorelCAD 1. Click the  buttons to find out more about the changes.

Corel SCRIPT Editor changes

-  The Corel SCRIPT Editor and the Corel SCRIPT Dialog Editor have been combined into one application, making it even easier to include custom dialog boxes in your scripts.
-  Corel SCRIPT commands, functions and statements can be continued over multiple lines by using two backslashes (\\).
-  In addition to line labels, Corel SCRIPT has added support for line numbers.
-  Scripts can now be assigned in the Corel SCRIPT Editor to a menu command, a shortcut key, or a toolbar button.

-  The Corel SCRIPT Editor now uses color coding to make the script syntax easier to read and understand.
-  Each line in a script can now hold 1,024 characters (including spaces). In Corel SCRIPT 6, each line could only hold 255 characters.
-  You can terminate a script's execution by pressing CTRL+BREAK.
-  Corel SCRIPT now supports application commands and functions for the Corel SCRIPT Editor. This means that you can now create scripts to create and modify other scripts.
-  Scripts can be compiled into executable programs.

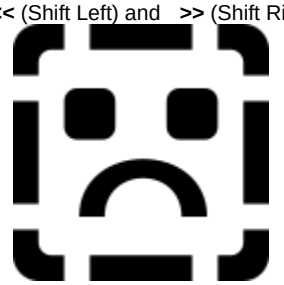
Modified Corel SCRIPT programming statements and functions

-  The **DECLARE** statement does not have to precede a **FUNCTION** or **SUB** statement.

-  The **EXIT** statement can now also be used with the **SELECT CASE** and **WHILE...WEND** structure.
-  The syntax for the **PRINT** and **WRITE** statements has changed.
-  The syntax for the **GETFILEBOX** statement has changed.

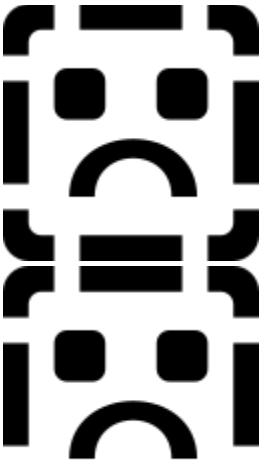
New Corel SCRIPT programming statements and functions

-  You can create dynamic dialog boxes in Corel SCRIPT. For a list of functions to use with the dynamic dialog boxes, click  .
-  Six new dialog controls for custom dialog boxes have been added to Corel SCRIPT: bitmap button, progress indicator, horizontal and vertical slider, and status control.

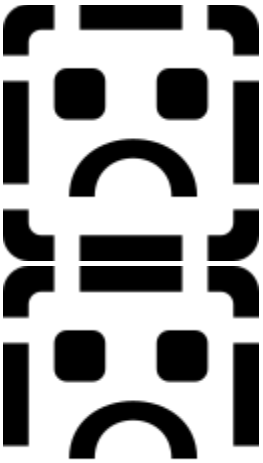
-  Four operators added to Corel SCRIPT: **EQV** and **IMP** operators (bitwise and logical), and **<<** (Shift Left) and **>>** (Shift Right) bitwise operators.
-  New Corel SCRIPT data type: VARIANT.
-  Three new Corel SCRIPT predefined dialog boxes have been added.

New Corel SCRIPT statements and functions

-  ADDFOL
-  ADDRESBMP
-  BEGINWAITCURSOR

-  BITMAPBUTTON
-  BUILDDATE
-  BUILDTIME
-  DEFINE
-  ENDWAITCURSOR
- FORMATDATE

- FORMATTIME
- GETCOLOR
- GETCOMMANDLINE
- GETCURRDATE
- GETCURRFOLDER
- GETDATEINFO

-  GETFOLDER
-  GETFONT
-  GETPROCESSINFO
-  GETSCRIPTFOLDER
-  GETTEMPFOLDER
-  GETTIMEINFO

- GETTYPE
- GETVERSION
- GOSUB
- RETURN
- HSLIDER
- INCLUDE

- PROGRESS
- REDIM
- REGISTRYQUERY
- SETCURRDATE
- SETCURRFOLDER
- SETEMPTY

-  STATUS
- STARTPROCESS
-  UNDEF
-  VSLIDER

Deleted Corel SCRIPT statements and functions

-  The **CURRFOLDER** statement has been replaced with the **GETCURRFOLDER** and **SETCURRFOLDER** statements.
-  The **CURRDATE** statement has been replaced with the **GETCURRDATE** and **SETCURRDATE** statements.

Changes in Corel SCRIPT from version 7 to 8

This topic outlines the changes to Corel SCRIPT since the release of version 7. Corel SCRIPT 7 was included with the following applications:

- CorelDRAW 7
- CorelDRAW 8 (8.232)
- Corel PHOTO-PAINT 7
- Corel PHOTO-PAINT 8 (8.232)
- Corel VENTURA 7

Click the  buttons to find out more about the changes.

Corel SCRIPT Editor changes

-  The Call Stack Window has been added. This dialog box displays a listing of the active functions in a script you are stepping through.

New (File menu)

Opens a new untitled script window.

 **Open (File menu)**

Opens the Run Script dialog box which is used to open a saved script file. The default folder and drive are shown, but you can open a script file in any drive or folder.

Close (File menu)

If a script window is active:

Closes the active script window. If your changes have not been saved, you are prompted to save before exiting.

If a dialog window is active:

Closes the active dialog window.

 **Save (File menu)**

If a script window is active:

Saves the script in the active window. If the script window is untitled, the Save As dialog box appears.

If a dialog window is active:

Saves the script in the window that launched the active dialog window. If the script has never been saved, the Save As dialog box appears.

 **Save As (File menu)**

If a script window is active:

Saves the script in the active window for the first time, or saves the script in the active window with a new name.

If a dialog window is active:

Saves the script in the window that launched the active dialog window for the first time or with a new name.

 **Save All (File menu)**

Saves all open scripts in the Corel SCRIPT Editor.

Export Dialog (File menu)

Closes all open script windows and dialog windows, and the Corel SCRIPT Editor. If you have not saved your scripts, you are prompted to save before exiting.

Print (File menu)

Opens the Print dialog box, which is used to print and set printing options. If the lines in your script are long, use the Print Setup command to set your printer to landscape orientation before printing a script.

Print Setup (File menu)

Opens the Print Setup dialog box, which allows you to choose the printer and printer options.

Print Direct (File menu)

Prints a copy of the active script using the current print settings. If the lines in your script are long, use the Print Setup command to set your printer to landscape orientation before printing a script.

Make EXE (File menu)

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

Make DLL (File menu)

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

Make CAO (File menu)

Creates a Corel Add-on (CAO) file from a script. Corel Add-ons are programs that add custom features to Corel applications. Third-party software developers are making Add-ons commercially available to meet the specialized needs of Corel users. Click  for more information.

Make CSB (File menu)

Creates a Corel SCRIPT Binary (CSB) file from a script. Unlike script files, binary files are compiled into machine language and don't have to be re-compiled each time they are run. Click  for more information.

Recently Opened Files (File menu)

Opens a recently opened Corel SCRIPT script file.

Exit (File menu)

Closes all open script and dialog windows, and the Corel SCRIPT Editor. If you have not saved your scripts, you are prompted to save before exiting.

 **Undo (Edit menu)**

Reverses actions performed during the current session. Use Undo after making a change you do not want implemented. Immediately after choosing Undo, the Redo command becomes available, allowing you to restore changes reversed by the Undo command.

 **Redo (Edit menu)**

Restores changes reversed by the Undo command. Redo becomes available immediately after you choose the Undo command.

Cut (Edit menu)

If a script window is active:

Cuts selected text from a script and places it on the Clipboard.

If a dialog window is active:

Cuts the selected dialog box, controls, or both from a dialog window and places them on the Clipboard as Corel SCRIPT statements.

Note

-  You can cut dialog control statements from a script and paste them into a dialog window.

Copy (Edit menu)

If a script window is active:

Copies selected text from a script and places it on the Clipboard.

If a dialog window is active:

Copies the selected dialog box, controls, or both from a dialog window and places them on the Clipboard as Corel SCRIPT statements.

Note

-  You can copy dialog control statements from a script and paste them into a dialog window.

 **Paste (Edit menu)**

If a script window is active:

Pastes text from the Clipboard at the insertion point. If you've selected text in a script, it is overwritten with the Clipboard contents.

If a dialog window is active:

Pastes Corel SCRIPT dialog control statements into the dialog box in the active dialog window from the Clipboard. A pasted control retains the original's label, identifier, size, and position attributes.

Note

-  If you try to paste Clipboard contents into a dialog window that contains invalid dialog definition statements, you are prompted to either ignore the statement that contains the invalid dialog definition statement, or create a new dialog box.

 **Delete (Edit menu)**

If a script window is active:

Deletes selected text from a script. If no further action has been performed, you can restore deleted text using the Undo command. Instead of deleting text, you can cut it. Cutting text transfers it to the Clipboard.

If a dialog window is active:

Deletes selected controls. If no further action has been performed, you can restore a deleted object using the Undo command. Instead of deleting a control, you can cut it. Cutting a control transfers it to the Clipboard as a Corel SCRIPT statement.

 **Duplicate (Edit menu)**

Adds a copy of selected control(s) to the active dialog box. By default, the copy is placed on top of the original, and offset down and to the right by 3 dialog units.

Note

-  The copied control(s) takes on the default control label and identifier. If you copied the control, the original's label and identifier are also copied.

Attributes (Edit menu)

Opens the Attributes dialog box for the selected dialog box, controls, or both. You can edit labels, identifiers, dialog and control size, and position attributes from the Attributes dialog box.

 **Select All (Edit menu)**

If a script window is active:

Selects all the text in the active script window.

If a dialog window is active:

Selects the dialog box in the active dialog window and every control in the dialog. The last control selected has a dotted line border.

Comment (Edit menu)

Places a **REM** statement at the beginning of a selected line in a script. Script lines beginning with REM statements are ignored during script execution.

Note

-  You can use this feature during debugging sessions to convert script syntax into remarks so they will be ignored during script execution.
-  Use the UnComment command to remove **REM** statements from selected lines in a script.

 **UnComment (Edit menu)**

Removes **REM** statements from selected lines in a script.

 **Cancel (Edit menu)**

Cancels an editing action in a dialog window, such as moving a control. It also unselects all selected controls including the dialog box.

 **Clear Output (Edit menu)**

Clears the Compiler Output Window in a script window of all information.

Find (Search menu)

Searches for specified text from the insertion point. You can set the search direction, match the case, and match entire words.

Replace (Search menu)

Searches for and replaces specified text from the insertion point. You can set the search direction, match the case, and match entire words.

Go To Line (Search menu)

Opens a dialog box that lets you choose the line to go to in a script. The status bar displays the line where the insertion point rests.

Next Error (Search menu)

Sends the insertion point to the next line in a script containing an error. The line where the insertion point is sent has the ⇨ symbol displayed in its left margin.

Before running this command, the Compiler Output window must display at least one error message and the insertion point must be in the script window.

 **Previous Error (Search menu)**

Sends the insertion point to the previous line in a script containing an error. The line where the insertion point is sent has the symbol displayed in its left margin.

Before running this command, the Compiler Output window must display at least one error message and the insertion point must be in the script window.

Go To Error (Search menu)

Sends the insertion point to the line containing the error. The insertion point must first be placed in an error line in the Compiler Output window. This command is also available with a right-mouse click.

Watch Window (View menu)

Opens and closes the Watch window. The Watch window monitors the value of specified variables in a script during a debugging session. The Watch window can be resized by clicking on a border and dragging.

Compiler Output Window (View menu)

Opens and closes the compiler output window. Before a script is run, it is compiled into an executable program. If errors during compilation occur, they are displayed in the Compiler Output window. The window updates each time a script is played or checked for syntax errors. The Compiler Output window can be resized by clicking on a border and dragging.

ToolBars (View menu)

Displays or hides toolbars, creates new toolbars, resets them to their original settings, adds or removes buttons and changes the button size.

Status Bar (View menu)

Toggles to display or the status bar found at the bottom of the Corel SCRIPT Editor. The status bar displays:

-  menu messages
-  the coordinates of the mouse pointer in a dialog box
- status of NUM lock and CAPS lock
-  the location of the insertion point in a script window in terms of lines and columns

Properties Bar (View menu)

Toggles the display of the Properties Bar in a dialog window. The Properties Bar displays attribute information about a selected control and can be used to edit dialog boxes and control attributes.

Controls List (View menu)

Toggles the display of the Controls List in a dialog window. The Controls List displays the order of the controls in a dialog box and can be used to re-order controls.

Run (Debug menu)

Runs the script in the active script window 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 with the Execute command.

If you're running a new script that has not yet been saved, you may be prompted to save before execution begins depending on your Corel SCRIPT Editor settings. See the How to section of the Corel Script help file for more information on automatically saving scripts before executing them.

Restart (Debug menu)

Stops and restarts script execution from the beginning. Variables are reset to their initial values. You can only use this command when you've paused script execution by stepping or using breakpoints.

Reset (Debug menu)

Ends script execution and resets variables to their initial values. You can only use this command when you've paused script execution by stepping or using breakpoints.

Step Into (Debug menu)

Executes a script line-by-line. The Step Into command also steps into functions and subroutines to execute lin-by-line.

Note

-  The line with the ➪ symbol in its left margin is the next line to execute.

Step Over (Debug menu)

Executes a script line-by-line. The Step Over command executes an entire procedure (a function or a subroutine) without stepping into the procedure's code.

Note

-  The line with the ➪ symbol in its left margin is the next line to execute.

Step Out (Debug menu)

Executes the remaining lines in a function or subroutine, and returns and stops at the line after the procedure call.

Note

-  The line with the ➪ symbol in its left margin is the next line to execute.

 **Run to Cursor (Debug menu)**

Runs the script in the active script window to the position of the insertion point. Since the insertion point acts as a breakpoint, using the Run to Cursor command is similar to using a breakpoint.

Execute (Debug menu)

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

If you are running a new script that has not yet been saved, you may be prompted to save before execution begins, depending on your Corel SCRIPT Editor settings. See the How to section of the Corel Script help file for more information on automatically saving scripts before executing them.

Toggle Breakpoint (Debug menu)

Sets a breakpoint on the line the insertion point is placed on. Choose the command again to clear the breakpoint. Script execution stops at a breakpoint, and you can have more than one breakpoint in a script.

Note

-  A line with a breakpoint has the symbol in its left margin.

 **Clear All Breakpoints (Debug menu)**

Clears all breakpoints in a script. Script execution stops at a breakpoint, and you can have more than one breakpoint in a script.

Note

-  A line with a breakpoint has the

 symbol in its left margin.

Check Syntax (Debug menu)

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

Double-click an error message's line number in the Compiler Output window to send the insertion point to the line containing the error. The line with the error has the ⇨ symbol in its left margin after double-clicking.

QuickWatch (Debug menu)

Opens the QuickWatch dialog box to monitor the value of a variable in the script. If the cursor is planted in a word when this command is issued, the word is place in the QuickWatch dialog box's Variable box. You can also use the QuickWatch window to add watches in the Watch Window. This option is only available when you're debugging (for example, stepping) a script.

Remove Watch (Watch menu)

Removes the selected variable(s) from the Watch window.

 **Remove All Watches (Watch menu)**

Removes all variable(s) from the Watch window.

Add Watch (Watch menu)

Adds a new variable to the Watch window. To use this command, plant the cursor the variable you want to add to the Watch window.

Dialog (Tools menu)

If your insertion point is in a dialog box definition, this command sends the script dialog box to a dialog window where the dialog statements can be edited. If the insertion point is not in a dialog box definition, a dialog window also opens, containing an empty dialog box that can be edited. The **BEGIN DIALOG** and **END DIALOG** statements are also inserted into the active script.

As you edit the dialog box, the script that launched the dialog window reflects the changes you make to the dialog box. For example, if you add a check box to the dialog box, a **CHECKBOX** statement is added to the script.

 Test Dialog (Tools menu)

Sets the dialog box in the active dialog window to test mode. In test mode, you can confirm whether a dialog box meets your requirements and functions properly. It is easier to test a dialog within the Corel SCRIPT Editor than to make it part of a script and test it by running a script. You cannot edit a dialog box in test mode. Press ESC to exit test mode. Pressing any push button or the Close Dialog button () also exits test mode.

Grid Settings (Tools menu)

Opens the Grid Settings dialog box, where you can set the following options for all dialog windows:

- Snap to Grid
- Grid spacing
- Grid display

Note

- Grid measurements are expressed in dialog units.
- To show the grid, enable Show Grid in the same dialog box.

Call Stack (Tools menu)

Opens the Call Stack window. This dialog box displays a listing of the active functions in a script you are stepping through.

Customize (Tools menu)

Opens the Customize dialog box. This dialog box lets you customize Corel SCRIPT Editor's toolbars, menu commands, and shortcut key assignments.

Options (Tools menu)

Opens a dialog box that modifies Corel SCRIPT Editor settings for script windows, colors, fonts, and INCLUDE files.

Corel Application Launcher

Launches installed Corel applications. You can also change the applications available on this button and add non-Corel applications to it.

Cascade (Window menu)

Layers script windows so each title bar is visible. To activate a script window, click the title bar. Minimized windows are arranged at the bottom of the Corel SCRIPT Editor window.

Tile Horizontally (Window menu)

Arranges the script and dialog windows horizontally in equal sizes to fit in the Corel SCRIPT Editor. Minimized windows are arranged at the bottom of the Corel SCRIPT Editor window.

Tile Vertically (Window menu)

Arranges the script and dialog windows horizontally in equal sizes to fit in the Corel SCRIPT Editor. Minimized windows are arranged at the bottom of the Corel SCRIPT Editor window.

 **Close All (Window menu)**

Closes all script and dialog windows. If your changes have not been saved, a confirmation message appears.

Arrange Icons (Window menu)

Arranges minimized script and dialog windows in the bottom-left corner of the Corel SCRIPT Editor.

Always on Top (Window menu)

Keeps the Corel SCRIPT Editor visible, even when another application is active. This is useful when you are debugging a script. Choose the command again to disable the setting.

Open document windows (Window menu)

Opens and activates script and dialog windows. The windows are listed in the order in which they were opened.

Help Topics (Help menu)

Opens the Corel SCRIPT Help Contents screen. From this screen, you can choose the type of Help you want. When you are in Help, click the Contents button to take you back to the opening screen.

What's This? command (Help menu)

Changes the cursor to the What's This cursor. Click this command, then click on an available menu command or another item on the screen to find out what it does.

About Corel SCRIPT Editor (Help menu)

Displays a dialog box with information about the version of Corel SCRIPT Editor you are running. Click the System Info button to open the System Info dialog box, which displays information about your system settings.

Selector (Control menu)

Sets the mouse pointer to the Selector state. In the Selector state, the mouse pointer appears as  and is used to select, resize, and drag and drop dialog controls. You can also select, move, and resize dialog boxes when in the Selector state.

In the Control state, the mouse pointer appears as and is used to create dialog controls. As soon as you create a tool, the mouse pointer resets to the Selector state. You can only set the mouse pointer to a Control state by clicking any of the dialog controls in the Control tool bar, or any control in the Control menu. The mouse pointer can only appear in a Control state when it is positioned over an active dialog box in a dialog window.

 **Text (Control menu)**

Sets the mouse pointer to the Control state so you can insert a Text control. In the Control state, the mouse pointer appears as +. You can insert a control by clicking, or by clicking and dragging in a dialog box.

 **Text Box (Control menu)**

Sets the mouse pointer to the Control state so you can insert a Text Box control. In the Control state, the mouse pointer appears as ⇨. You can insert a control by clicking, or by clicking and dragging in a dialog box.

 OK Button (Control menu)

Sets the mouse pointer to the Control state to insert an OK Button control. In the Control state, the mouse pointer appears as  . You can insert a control by clicking, or by clicking and dragging in a dialog box.

 Cancel Button (Control menu)

Sets the mouse pointer to the Control state so you can insert a Cancel Button control. In the Control state, the mouse pointer appears as  . You can insert a control by clicking, or by clicking and dragging in a dialog box.

Push Button (Control menu)

Sets the mouse pointer to the Control state so you can insert a Push Button control. In the Control state the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

 **Bitmap Button (Control menu)**

Sets the mouse pointer to the Control state so you can insert a Bitmap Button control. In the Control state the mouse pointer appears as ⇨. You can insert a control by clicking, or by clicking and dragging in a dialog box.

Option Button (Control menu)

Sets the mouse pointer to the Control state so you can insert an Option Button control. In the Control state, the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

Check Box (Control menu)

Sets the mouse pointer to the Control state so you can insert a Check Box control. In the Control state, the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

List Box (Control menu)

Sets the mouse pointer to the Control state so you can insert a List Box control. In the Control state, the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

 Drop-down List Box (Control menu)

Sets the mouse pointer to the Control state so you can insert a Drop-down List Box control. In the Control state, the mouse pointer appears as  . You can insert a control by clicking or clicking and dragging in a dialog box.

Combo Box (Control menu)

Sets the mouse pointer to the Control state so you can insert a Combo Box control. In the Control state, the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

 Drop-down Combo Box (Control menu)

Sets the mouse pointer to the Control state so you can insert a Drop-down Combo Box control. In the Control state, the mouse pointer appears as  . You can insert a control by clicking, or by clicking and dragging in a dialog box.

Group Box (Control menu)

Sets the mouse pointer to the Control state so you can insert a Group Box control. In the Control state, the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

Spin Control (Control menu)

Sets the mouse pointer to the Control state so you can insert a Spin control. In the Control state, the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

 **Horizontal Slider (Control menu)**

Sets the mouse pointer to the Control state so you can insert a horizontal slider control. In the Control state, the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

Vertical Slider (Control menu)

Sets the mouse pointer to the Control state so you can insert a vertical slider control. In the Control state, the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

Progress Indicator (Control menu)

Sets the mouse pointer to the Control state so you can insert a Progress Indicator control. In the Control state, the mouse pointer appears as ⇨. You can insert a control by clicking, or by clicking and dragging in a dialog box.

 Image List Box (Control menu)

Sets the mouse pointer to the Control state so you can insert a Image List Box control. In the Control state the mouse pointer appears as  . You can insert a control by clicking, or by clicking and dragging in a dialog box.

Image (Control menu)

Sets the mouse pointer to the Control state so you can insert a Image control. In the Control state the mouse pointer appears as ➪. You can insert a control by clicking, or by clicking and dragging in a dialog box.

 Help Button (Control menu)

Sets the mouse pointer to the Control state so you can insert a Help button control. In the Control state the mouse pointer appears as  . You can insert a control by clicking, or by clicking and dragging in a dialog box.

Status control (Control menu)

Sets the mouse pointer to the Control state so you can insert a status control. In the Control state the mouse pointer appears as ⇨. You can insert a control by clicking, or by clicking and dragging in a dialog box.

Corel SCRIPT custom dialog box

Click Edit, Attributes to change the custom dialog box's properties.

Align Control, Left (Layout menu)

Aligns selected controls along their left edge. More than one control must be selected to activate this command.

Note

- The last control you select maintains its position; all other selected controls move to align with this control.

Align Control, Right (Layout menu)

Aligns selected controls along their right edge. More than one control must be selected to activate this command.

Note

- The last control you select maintains its position; all other selected controls move to align with this control.

Align Control, Top (Layout menu)

Aligns selected controls along their top edge. More than one control must be selected to activate this command.

Note

- The last control you select maintains its position; all other selected controls move to align with this control.

Align Control, Bottom (Layout menu)

Aligns selected controls along their bottom edge. More than one control must be selected to activate this command.

Note

- The last control you select maintains its position; all other selected controls move to align with this control.

Distribute, Horizontal (Layout menu)

Spaces selected controls evenly across a dialog box. The selected controls are spaced evenly between the leftmost and rightmost borders of the selected controls.

Note

- More than two controls must be selected for this command to be active.

Distribute, Vertical (Layout menu)

Spaces selected controls evenly down a dialog box. The selected controls are spaced evenly between the topmost and bottommost borders of the selected controls.

Note

- More than two controls must be selected for this command to be active.

Center in Dialog, Horizontal (Layout menu)

Centers selected control(s) horizontally in a dialog box. The controls are centered horizontally based on the leftmost and rightmost borders of the selected controls.

Center in Dialog, Vertical (Layout menu)

Centers selected control(s) vertically in a dialog box. The controls are centered vertically based on the topmost and bottommost borders of the selected controls.

Make Same Size, Width (Layout menu)

Resizes selected controls to make them the same width. The last control you select maintains its width; all other selected controls resize to this control.

Make Same Size, Height (Layout menu)

Resizes selected controls to make them the same height. The last control you select maintains its height; all other selected controls resize to this control.

Make Same Size, Both (Layout menu)

Resizes selected controls to make them the same width and height. The last control you select maintains its width and height; all other selected controls resize to this control.

Size to Content (Layout menu)

- Resizes a control(s) to fit its label.

Note

- ⇨ You can use this command on option buttons, check boxes, push buttons, and text.

Snap to Grid (Layout menu)

Toggles to enable or disable Snap to Grid in all the dialog windows. If you are moving and resizing controls in a dialog box using the mouse, enabling Snap to Grid can help to accurately align and position objects along a grid. It also allows you to draw precisely-sized objects.

Note

- Snap to Grid can be enabled without showing the grid.

Displays a description for the selected Corel SCRIPT script. The description is stored in a script's first two lines as remark (REM) statements.

Dialog and Control Attributes dialog box

The following attributes for dialog boxes and controls can be edited in this dialog box:

- Width (in dialog units)
- Height (in dialog units)
- X (left border position)
- Y (top border position)
- Center Dialog option
- Value
- Text
- Comment
- Dynamic dialog box setting

The attributes become parameters when the dialog and the controls are saved as Corel SCRIPT statements. For reference information about a dialog control's attributes, click Related Topics.

Notes

- If you specify a text attribute with a string it must be enclosed with quotes. If you specify a text attribute with a variable, do not use quotes. For example, a check box has a text attribute. If you wanted to set it to DOG, you could either type the string "DOG" into the check box's text attribute box or create a string variable, assign it DOG, and type the string variable's name into the check box's text attribute box.
- If the Center Dialog option is enabled when a dialog box is selected, the X and Y number boxes are grayed and cannot be edited. By default, a new dialog has the Center Dialog check box enabled.
- You can select more than one control and open the Attributes dialog box. In this case, the title bar of the Attributes box displays "Multiple Selection Attributes", and allows you to edit all the selected controls at once. For example, if you select a check box and option button, you can set them both to have the same height and width. In some multiple selection cases, attribute options may be grayed and not available to edit.
- To get information on an option in a dialog box, click ?, then click the option.

{button ,AL(`corel_script_dialog_control;corel_script_dialog_control_dynamic;;;;',0,"Defaultoverview",)} Related Topics

Provides a space for you to enter the width of the selected dialog box or control(s) in dialog units.

Provides a space for you to enter the height of the selected dialog box or control(s) in dialog units.

Provides a space for you to enter the distance in dialog units from the inside of the dialog box's left border to the left side of the selected control(s).

For dialog boxes, provides a space for you to enter the distance in dialog units from the dialog box's left border to the left side of the monitor's display area.

Provides a space for you to enter the distance in dialog units from the bottom of the dialog box's title bar to the top of the selected control(s).

For dialog boxes, provides a space for you to enter the distance in dialog units from the dialog box's top border to the top side of the monitor's display area.

Centers the dialog box on the monitor during a script run. Ignores the entries in the X and Y boxes when enabled.

Provides a space for you to set a dialog box or dialog control(s) attribute. The attribute is dependent on the control(s) selected, or if a dialog box is selected. Click the Help button in this dialog box for a list of attributes.

Provides a space for you to set a dialog box or dialog control(s) attribute. The attribute is dependent on the control(s) selected, or if a dialog box is selected. Click the Help button in this dialog box for a list of attributes.

Provides a space for you to enter a comment (remark) for the selected dialog box or dialog control(s). The comment appears in the script to the right of the statement and is preceded by an apostrophe.

Provides a space for you to set a dialog box or dialog control(s) attribute. The attribute is dependent on the control(s) selected, or if a dialog box is selected. Click the Help button in this dialog box for a list of attributes.

Set this option to make your custom dialog box dynamic. As a dynamic dialog box, its content can change based on user input.

Go To dialog box

Specifies the line to move the insertion point to in the active script window. To get information on an option in a dialog box, click  , then click the option.

Specifies the line to move the insertion point to in the active script window.

QuickWatch dialog box

The QuickWatch dialog box can be used to help you to debug your scripts in the following ways:

- Monitor the value of a variable in the script. As you step through a script, the value in the QuickWatch window changes as the variable changes.
- Add variables to the Watch window.
- Change the value of a variable being watched.
- Show the data subtype of a variant being watched. You can also change the variant's subtype using this dialog box. Click for more information about variants.

Note

- To get information about an option in a dialog box, click , then click the option.

Displays the variable or expression to watch.

Displays the current value of the specified variable or expression. You can change a variable's value during script debugging by typing a value.

Adds the specified variable or expression to the Watch window. Also opens the Watch window, if it is closed.

Specifies the data type of the variable being watched.

Changes the value of the specified variable using the value in the Value box.

Click to change the data subtype of a variant being watched. Before clicking, select a subtype from the list box. Click  for more information about variants.

Displays the data subtype when a variant is being watched. Click  for more information about variants.

Options dialog box

The Options dialog box modifies the Corel SCRIPT Editor settings for script windows, colors, fonts, compiling, and files.

Editor tab

Sets tab and autosave options in the Corel SCRIPT Editor.

Colors tab

Sets text and background display colors in the Corel SCRIPT Editor.

Font tab

Sets display and print fonts in the Corel SCRIPT Editor.

Folders tab

Sets the folders that are searched for INCLUDE files during a script compile. See the **INCLUDE** statement for more information.

Environment tab

Sets whether to require explicit declaration of all variables. For more information about explicit declaration, click .

Note

- To get information on an option in a dialog box, click , then click the option.

Enable to require explicit declaration of all variables. For more information about explicit declaration, click ⇨.

Enter a value to test the **GETCOMMANDLINE** function.

Specifies the tab spacing to use in scripts. Tabs can help you to format a script so it is easier to read and debug.

Specifies whether scripts should be saved before they are executed or run.

Specifies whether a new script line has the same number of tabs at its left side as the previous line after the ENTER key is pressed.

Enable to display color codes in script. Color codes help to differentiate the different items in a script. The items are listed in the Set Text Color For list box.

List the available text colors for the text items specified in the Set Colors For list box.

List the available background colors for the text items specified in the Set Colors For list box.

List the text items to set color attributes for.

Displays a color sample of the selected item in the Set Colors For list box.

Resets the text item specified in the Set Colors For dialog box to its original Corel SCRIPT Editor color.

Resets all text items in the Set Colors For dialog box to their original Corel SCRIPT Editor colors.

Displays the name of the font used in script windows.

Displays the point size of the font used in script windows.

Displays a sample of the font used in script windows.

Resets the font used in script windows to the original Corel SCRIPT Editor setting.

Opens a dialog box to change the font properties used in script windows.

Lists the folders to search for INCLUDE files. The folders are searched in the order in which they are listed. See the **INCLUDE** statement for more information.

Removes selected folders from the folder search list.

Opens a dialog box used to add folders to the folder search list.

Specifies the type of folders displayed in the folder search list. Currently, you can only display folders that contain script files (.CSC). See the **INCLUDE** statement for more information.

Specifies a folder to add to the folders list. Drive information should also be included.

Opens a dialog box used to select folders for the folders list.

Grid Setting dialog box

The Grid Setting dialog box sets options to help you to accurately align, size, and position controls.

Show Grid

Displays a series of dotted horizontal and vertical lines. Working with the grid on makes it easier to accurately align and position controls.

The dots appear where the lines intersect. The Horizontal and Vertical number boxes set the line spacing.

Snap to Grid

Enabling Snap to Grid forces the mouse pointer to stay on the underlying grid when you insert, move, and resize controls.

- Snap to Grid can be enabled without showing the grid.

Note

- To get information on an option in a dialog box, click , then click the option.

Displays a series of dotted horizontal and vertical lines. Working with the grid on makes it easier to accurately align and position controls. The setting is applied to all dialog windows.

The dots appear where the lines intersect. The Horizontal and Vertical number boxes set the line spacing.

Enabling Snap to Grid forces the mouse pointer to stay on the underlying grid when you insert, move, and resize controls. The setting is applied to all dialog windows.

- You can enable Snap to Grid without showing the grid.

Provides a space for you to enter the grid's horizontal line spacing in dialog units.

Provides a space for you to enter the grid's vertical line spacing in dialog units.

Opens detailed online Help for this dialog box.

Closes this dialog box without saving any changes you have made.

Closes this dialog box and saves any changes you have made.

Applies the changes made to the tab sheet.

Provides a space for you to enter the distance in dialog units from the inside of the dialog box's left border to the left side of the selected control.

Provides a space for you to enter the distance in dialog units from the bottom of the dialog box's title bar to the top of the selected control.

Provides a space for you to enter the width of the selected dialog control in dialog units.

Provides a space for you to enter the height of the selected dialog control in dialog units.

Provides a space for you to select the dialog control's tab order number. The lowest number you can enter is 1.

#Specifies a string variable that identifies the selected dynamic dialog control.

Displays the dialog controls in the dialog box in their tab order.

Moves the selected control up one position in the dialog tab order.

Moves the selected control down one position in the dialog tab order.

This is a Corel SCRIPT script or dialog window. Script windows are used to edit scripts while dialog windows are used to edit custom dialog boxes for Corel SCRIPT.

This window displays a Corel SCRIPT script. The window is called the script window.

Provides a space to type the name of a variable to add to the Watch Window. Press Enter to add the variable to the Watch Window.

Adds the variable in the Watch box to the Watch window.

Removes the selected variable(s) from the Watch window.

This window displays the current value for each monitored variable and the procedure where the variable is located. The window is called the Watch window.

This window displays the current value for each monitored variable and the procedure where the variable is located. The window is called the Watch window.

Type a variable name in the text box to add the variable to the Watch box. Select a variable in the Watch box and click the button to remove it.

This window displays errors after a script has been run or checked for syntax errors. The window is called the Compiler Output window.

This is the dialog window. The dialog box displayed in the dialog window is a graphical representation of Corel SCRIPT statements. Working in this window is similar to using a drawing or painting application: dialog controls are graphic objects which can be inserted, moved, resized, and aligned in a dialog box.

Reduces the Corel SCRIPT Editor or a script or dialog window to an icon.

Enlarges the Corel SCRIPT Editor to fit the screen or Restores the Corel SCRIPT Editor or a script or dialog window to it previous size and location.

Closes a script or dialog window.

Reduces the Corel SCRIPT Editor or a script or dialog window to an icon.

Enlarges the Corel SCRIPT Editor to fit the screen or Restores the Corel SCRIPT Editor or a script or dialog window to it previous size and location.

Closes a script or dialog window.

Drag the title bar to move the window. Double click the title bar to maximize the window.

status bar whats this

This is the Status bar which provides a variety of functions. It displays items:

- mouse pointer coordinates in dialog units when positioned over a dialog box in a dialog window.
- status bar messages when the mouse pointer is not positioned over a dialog box.
- Corel SCRIPT Editor status information such as NUM and CAP lock settings. You can also change the status by clicking on these settings.

The Status bar can be customized and can be displayed at the top or the bottom of the Corel SCRIPT Editor or can be hidden. Right-click on the Status bar to change its settings.

ANGLECONVERT function

ANGLECONVERT(x, y, z)

Converts a number from one angle measurement to another.

Parameter	Description
x	Any number from 1 to 5 that indicates the unit of measurement from which to convert. 1 = degrees 2 = radians 3 = gradients 4 = Corel PHOTO-PAINT degrees (tenths of a degree) 5 = CorelDRAW degrees (millionths of a degree)
y	Any number from 1 to 5 that indicates the unit of measurement to convert to. 1 = degrees 2 = radians 3 = gradients 4 = Corel PHOTO-PAINT degrees (tenths of a degree) 5 = CorelDRAW degrees (millionths of a degree)
z	Any numeric expression specifying the value to be converted.

Example

x_rads = ANGLECONVERT(1, 2, 90)

The above example converts 90 degrees to radians. The variable **x_rads** equals 1.57142857142932.

{button ,AL(`include;cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

FROMCENTIMETERS function

FROMCENTIMETERS(x)

Converts a numeric value from centimeters to tenths of a micron.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.CreateRectangle FROMCENTIMETERS(8), FROMCENTIMETERS(-5), FROMCENTIMETERS(0), FROMCENTIMETERS(2.5), FROMCENTIMETERS(0.75)

This CorelDRAW command would create a rectangle 7.5 by 8 centimeters. The rectangle's top left corner coordinate is -5, 8 centimeters relative to the center of the page, and the corners are 0.75 centimeters in diameter.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

FROMCICEROS function

FROMCICEROS(x)

Converts a numeric value from ciceros to tenths of a micron.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.CreateRectangle FROMCICEROS(18), FROMCICEROS(-12), FROMCICEROS(8), FROMCICEROS(6), FROMCICEROS(1.5)

This CorelDRAW command would create a rectangle 18 by 10 ciceros. The rectangle's top left corner coordinate is -12, 18 ciceros relative to the center of the page, and the corners are 1.5 ciceros in diameter.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

FROMDIDOTS function

FROMDIDOTS(x)

Converts a numeric value from didots to tenths of a micron.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.CreateRectangle FROMDIDOTS(50), FROMDIDOTS(-70), FROMDIDOTS(0), FROMDIDOTS(30), FROMDIDOTS(20)

This CorelDRAW command would create a rectangle 100 by 50 didots. The rectangle's top left coordinate is -70, 50 didots relative to the center of the page, and the corners are 20 didots in diameter.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

FROMINCHES function

FROMINCHES(x)

Converts a numeric value from inches to tenths of a micron.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.CreateRectangle FROMINCHES(3), FROMINCHES(-2), FROMINCHES(0), FROMINCHES(1), FROMINCHES(0.25)

This CorelDRAW command would create a rectangle 3 by 3 inches. The rectangle's top left coordinate is -2, 3 inches relative to the center of the page, and the corners are 0.25 inches in diameter.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

FROMPICAS function

FROMPICAS(x)

Converts a numeric value from picas to tenths of a micron.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.CreateRectangle FROMPICAS(18), FROMPICAS(-12), FROMPICAS(8), FROMPICAS(6), FROMPICAS(2)

This CorelDRAW command would create a rectangle 18 by 10 picas. The rectangle's top left coordinate is -12, 18 picas relative to the center of the page, and the corners are 2 picas in diameter.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

FROMPOINTS function

FROMPOINTS(x)

Converts a numeric value from points to tenths of a micron.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.CreateRectangle FROMPOINTS(210), FROMPOINTS(-140), FROMPOINTS(90), FROMPOINTS(70), FROMPOINTS(1.75)

This CorelDRAW command would create a rectangle 210 by 140 points. The rectangle's top left corner coordinate is -140, 210 points relative to the center of the page, and the corners are 1.75 points in diameter.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

LENGTHCONVERT function

LENGTHCONVERT(x, y, z)

Converts a number from one length measurement to another.

Parameter	Description
x	Any number from 1 to 7 that indicates the unit of measurement from which to convert. 1 inches 2 centimeters 3 points 4 Ciceros 5 didots 6 picas 7 CorelDRAW and VENTURA units (tenths of a micron)
y	Any number from 1 to 7 that indicates the unit of measurement to convert to: 1 inches 2 centimeters 3 points 4 Ciceros 5 didots 6 picas 7 CorelDRAW and VENTURA units (tenths of a micron)
z	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

x_microns = LENGTHCONVERT(1, 7, 1)

The above example converts one inch to tenths of a micron. The variable **x_microns** equals 254,000.

{button ,AL(`include;cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

TOCENTIMETERS function

TOCENTIMETERS(x)

Converts a numeric value from tenths of a micron to centimeters.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.GetPosition (x,y)

xCM = TOCENTIMETERS (x)

yCM = TOCENTIMETERS (y)

In this CorelDRAW example, **xCM** and **yCM** are set to the X and Y coordinates of the selected object in centimeters.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

TOCICEROS function

TOCICEROS(x)

Converts a numeric value from tenths of a micron to ciceros.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.GetPosition (x,y)

xCiceros = TOCICEROS (x)

yCiceros = TOCICEROS (y)

In this CorelDRAW example, **xCiceros** and **yCiceros** are set to the X and Y coordinates of the selected object in ciceros.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

TODIDOTS function

TODIDOTS(x)

Converts a numeric value from tenths of a micron to didots.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.GetPosition (x,y)

xDidots = TODIDOTS (x)

yDidots = TODIDOTS (y)

In this CorelDRAW example, **xDidots** and **yDidots** are set to the X and Y coordinates of the selected object in didots.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

TOINCHES function

TOINCHES(x)

Converts a numeric value from tenths of a micron to inches.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.GetPosition (x,y)

xInch = TOINCHES (x)

yInch = TOINCHES (y)

In this CorelDRAW example, **xInch** and **yInch** are set to the X and Y coordinates of the selected object in inches.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

TOPICAS function

TOPICAS(x)

Converts a numeric value from tenths of a micron to picas.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.GetPosition (x,y)

xPica = TOPICAS (x)

yPica = TOPICAS (y)

In this CorelDRAW example, **xPica** and **yPica** are set to the X and Y coordinates of the selected object in picas.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

TOPOINTS function

TOPOINTS(x)

Converts a numeric value from tenths of a micron to points.

Parameter	Description
x	Any numeric expression specifying the value to be converted.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW and Corel VENTURA.

Example

.GetPosition (x,y)

xPoint = TOPOINTS (x)

yPoint = TOPOINTS (y)

In this CorelDRAW example, **xPoint** and **yPoint** are set to the X and Y coordinates of the selected object in points.

{button ,AL(`cs_converts;;;;;',0,"Defaultoverview",)} Related Topics

GETCOLOR statement and function

Statement syntax
GETCOLOR Red, Green, Blue

Function syntax
ReturnValue = GETCOLOR (Red, Green, Blue)

Displays a standard Windows Color dialog box and returns color setting values from the RGB color model (Red, Green, Blue).

Return Value

The GETCOLOR function returns one of the following values

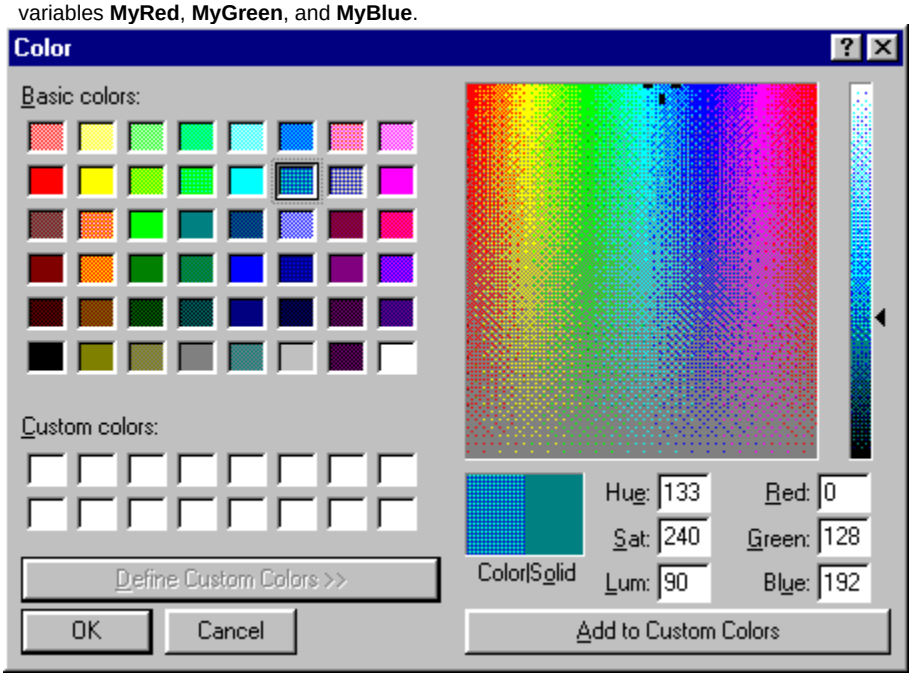
- TRUE (-1)⇨the Cancel button was not clicked
- FALSE (0)⇨the Cancel button was clicked

Parameter	Definition
Red	Lets you specify the numeric variable that is passed the Red setting of the selected color (0 - 255). You can also use this variable to set an initial value.
Green	Lets you specify the numeric variable that is passed the Green setting of the selected color (0 - 255). You can also use this variable to set an initial value.
Blue	Lets you specify the numeric variable that is passed the Blue setting of the selected color (0 - 255). You can also use this variable to set an initial value.

Example

GETCOLOR MyRed%, MyGreen%, MyBlue%

The above example displays the following dialog box and returns the RGB color settings for the selected color to the numeric variables **MyRed**, **MyGreen**, and **MyBlue**.



{button ,AL(`cs_ui_statements;;;;;',0,"Defaultoverview",)} **Related Topics**

GETFILEBOX function

GETFILEBOX(Filter, Title, Type, DefFile, DefExt, DefFol, BtnName**)**

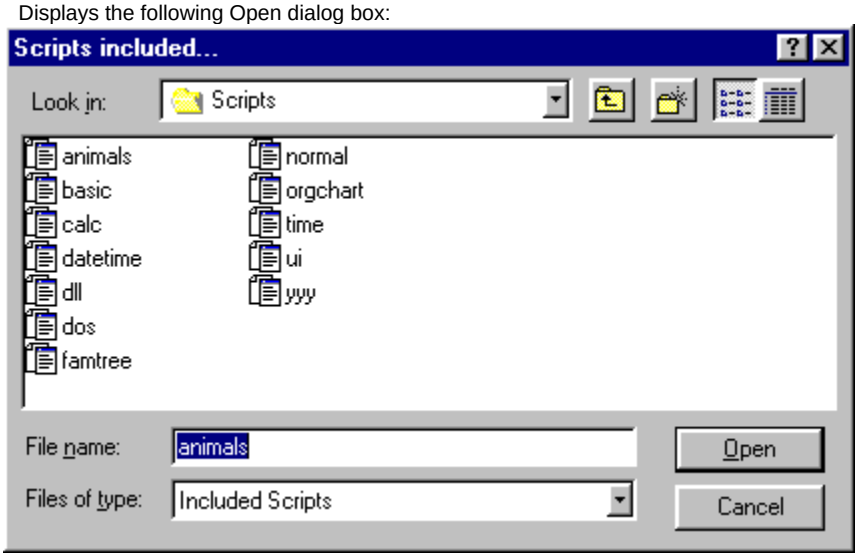
This function displays a standard Windows File Open or File Save As dialog box. Both dialog boxes allow users to choose a file from the file system. The **GETFILEBOX** function returns the selected filename and its full path, or an empty string if the user chooses Cancel. The **GETFILEBOX** statement by itself does not open or save a file; it only returns a string corresponding to the selected file.

Parameter	Definition
Filter	String expression specifying the filters to use in the dialog box. For the Open dialog box, the filters are listed in the Files of Type list box. For the Save As dialog box, the filters are listed in the Save as Type list box. Filters are specified in two parts. The first part is the text that appears in the list box, and the second part is the actual filter extension. The parts are separated by the | character (do not use spaces before or after the | characters). To separate multiple filters, use the | character. See the example below for more information.
Title	String expression specifying the title to display in the dialog box. If not specified, "Open" is displayed for an Open dialog box and "Save As" is displayed for a Save As dialog box.
Type	Numeric expression specifying the type of dialog box to display: 0 File Open dialog box (default if omitted) 1 File Save dialog box
DefFile	String expression specifying the text to display in the File name text box of the dialog box. If not specified, the text box is empty.
DefExt	String expression specifying the default extension to append to a File name if the user omits the extension.
DefFol	String expression specifying the default folder used by the dialog box. If not specified, or the specified folder does not exist, the current folder is used.
BtnName	String expression specifying a button name to override the Open or Save button in the dialog box. If not specified, the button's name remains unchanged.

Example

```
SETCURRFOLDER = "c:\COREL50\DRAW\samples" 'set the current folder
Filename$=GETFILEBOX("Included Scripts|*.csc|All Files|*.*", "Scripts included...", 0,"animals")
```

Displays the following Open dialog box:



{button ,AL(`cs_ui_statements;chfolder;;;;',0,"Defaultoverview",)} **Related Topics**

GETFOLDER function

GETFOLDER (InitFolder**)**

This function displays a Windows Choose Folder dialog box. The Choose Folder dialog box returns the folder and path a user chooses as a string.

Parameter	Definition
InitFolder	String expression specifying the default path and folder to display in the dialog box. If not specified, the active folder is used.

Note

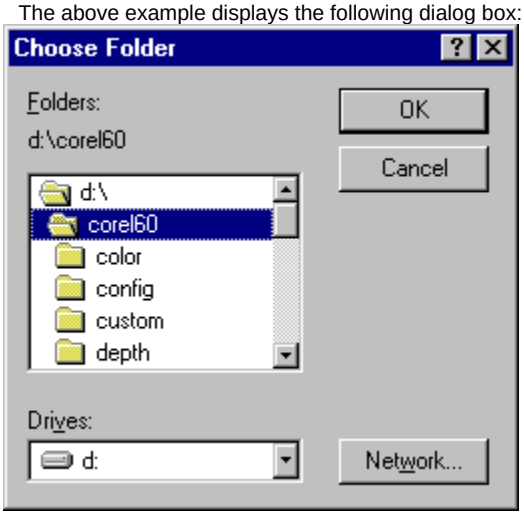
- If the Cancel button is clicked, an empty string is returned.

Example

NewFolder$ = GETFOLDER("D:\Corel60")

SETCURRFOLDER = NewFolder$

The above example displays the following dialog box:



The selected folder is passed to the string variable **NewFolder**. The **SETCURRFOLDER** statement sets the current folder to the folder name passed to **NewFolder**.

{button ,AL(`cs_ui_statements;chfolder;currfolder;;;',0,"Defaultoverview",)} Related Topics

GETFONT statement and function

Statement syntax
GETFONT FaceName, PointSize, Weight, Italic, Underline, StrikeOut, Red, Green, Blue

Function syntax
ReturnValue = GETFONT (FaceName, PointSize, Weight, Italic, Underline, StrikeOut, Red, Green, Blue)

Displays a standard Windows Font dialog box and returns the selected font settings.

Return Value

The GET font function returns one of the following values:

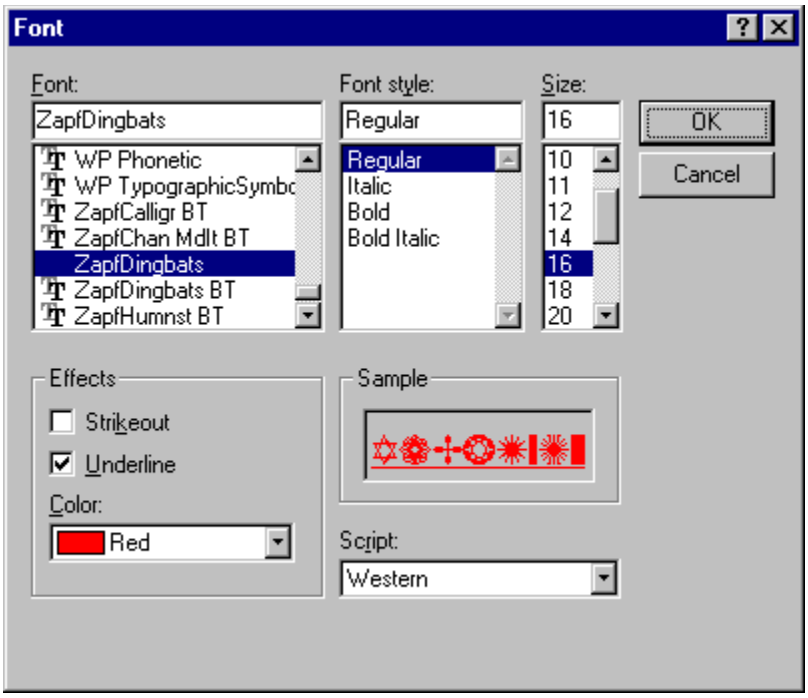
- TRUE (-1)⇨the Cancel button was not clicked
- FALSE (0)⇨the Cancel button was clicked

Parameter	Definition
FaceName	Lets you specify a string variable that is passed the name of the selected font. You can also use this variable to set an initial value.
PointSize	Lets you specify a numeric variable that is passed the font size in points. You can also use this variable to set an initial value. This parameter uses non-fractional values
Weight	Lets you specify a numeric variable that is passed the font's weight setting (number of inked pixels per 1000 pixels). Common values and their corresponding names include: 100 Thin 200 Extra Light, Ultra Light 300 Light **400 Normal, Regular** 500 Medium 600 Semi Bold, Demi Bold **700 Bold** 800 Extra Bold, Ultra Bold 900 Black, Heavy Most Windows fonts only use two weight settings: 400 (Normal) and 700 (Bold).
Italic	Lets you specify a numeric variable that is passed the font's italic setting: TRUE (-1) if this setting is enabled; FALSE (0) otherwise.
Underline	Lets you specify a numeric variable that is passed the font's underline setting: TRUE (-1) if this setting is enabled; FALSE (0) otherwise.
StrikeOut	Lets you specify a numeric variable that is passed the font's strike out setting: TRUE (-1) if this setting is enabled; FALSE (0) otherwise.
Red	Lets you specify the numeric variable that is passed the Red (RGB color model) setting of the selected font's color (0 - 255). You can also use this variable to set an initial value.
Green	Lets you specify the numeric variable that is passed the Green (RGB color model) setting of the selected font's color (0 - 255). You can also use this variable to set an initial value.
Blue	Lets you specify the numeric variable that is passed the Blue (RGB color model) setting of the selected font's color (0 - 255). You can also use this variable to set an initial value.

Example

GETFONT FN, PS, Wt, Italic, UL, SO, R, G, B

The above example displays the following dialog box and returns the font settings the variables specified.



{button ,AL(`cs_ui_statements;;;;;',0,"Defaultoverview",)} **Related Topics**

INPUTBOX function

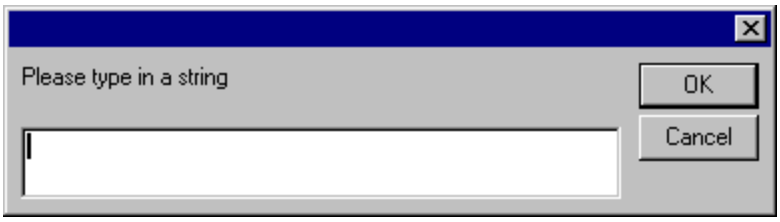
INPUTBOX(prompt$)

Displays a simple dialog box where you can enter a string that is returned to a script. The dialog box has OK and Cancel buttons. If the Cancel button is chosen, an empty string is returned.

Parameter	Definition
prompt$	String expression that appears in the dialog box above the edit box.

Example

MyString$ = INPUTBOX("Please type in a string")



User input is returned to the variable **MyString$.**

{button ,AL(`cs_ui_statements;textbox;text;;;',0,"Defaultoverview",)} Related Topics

MESSAGE statement

Message anyVariable

Displays a dialog box that contains a specified message and an OK button. This statement doesn't return any value to a running script, but can provide the script user with information during script execution.

Parameter	Definition
anyVariable	Any numeric or string expression to display in the message box. Numbers and dates are displayed as their string representations.

Example

```
x$="Hello." + CHR(13)     'CHR(13) is a return character
MESSAGE x$ + "What a nice day."
```

Note

- See the Corel SCRIPT character map for a list of character codes.

{button ,AL(`cs_ui_statements;textbox;text;chr;;',0,"Defaultoverview",)} Related Topics

MESSAGEBOX function

MESSAGEBOX(prompt, title, option**)**

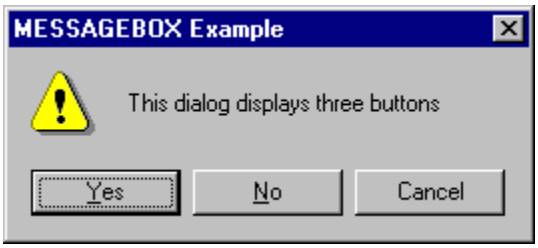
Displays a message box with a specified message and user-specified buttons and icons. MESSAGEBOX returns a value representing the button that was used to close it to the script.

Parameter	Definition
prompt	String expression to display in the box.
title	String expression to display in the message box caption.
option	A numeric expression representing the type of buttons to include in the box and an icon (if any) to appear beside the message. The **option** value is set using the OR (or +) operator for multiple buttons (see example): **Button Type:** 0 OK only (used by default if a button is not specified) 1 OK/Cancel 2 Abort/Retry/Ignore 3 Yes/No/Cancel 4 Yes/No 5 Retry/Cancel **Icon Type** (click hot spot for an example)**:** 0 No icon 16 Stop 32 Question 48 Exclamation 64 Information

Returns	Button pressed
1	OK
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

Example

retval% = MESSAGEBOX("This dialog displays three buttons", "MESSAGEBOX Example", 3 OR 48)



Note

- Pressing the Close Dialog button (⇨) is the same as pressing the Cancel button; both return 2.

{button ,AL(`cs_ui_statements;textbox;text;;;',0,"Defaultoverview",)} Related Topics

To assign a Corel SCRIPT script to a shortcut key

From your Corel application or from the Corel SCRIPT Editor:

1. Click Tools, Customize.
2. Click Keyboard.
3. In the Commands box, double-click the Application Scripts folder or the General Scripts folder.

   The Application Scripts folder contains scripts that send instructions to the Corel application you are customizing. The General Scripts folder contains scripts that are not application specific.

   Based on a typical Corel installation, application scripts reside in the **C:\COREL*CorelSuite**application*\SCRIPTS** folder, where ***CorelSuite*** refers to the Corel products installed and ***application*** refers to the Corel application's folder. For example, CorelDRAW 7 scripts may reside in **C:\COREL\DRAW70\DRAW\SCRIPTS** folder and Corel VENTURA 7 scripts may reside in the **C:\COREL\VENTURA7\VENTURA\SCRIPTS** folder.

   General scripts normally reside in the **C:\COREL*CorelSuite*\SCRIPTS** folder.
4. Click the script.
5. Click the Press new shortcut key box.
6. Press the keyboard combination you want to assign to the command. To make a correction, press the Backspace key.

   You can have up to four layers of keystrokes. For example, the key combination CTRL+ALT+1,2,3,4 is accomplished by holding down the CTRL and ALT keys, then pressing the 1,2,3, and 4 keys in succession.

Note

- You can also assign a Corel SCRIPT Binaries to a shortcut key by using the above procedure.
- To have accelerator conflicts resolved automatically, enable Go to conflict on assign.

{button ,AL(`cs_custom;;;;;',0,"Defaultoverview",)} Related Topics

To assign a Corel SCRIPT script to a menu

From your Corel application or from the Corel SCRIPT Editor:

1. Click Tools, Customize.
2. Click Menu.
3. In the Commands box, double-click the Application Scripts folder or the General Scripts folder.

   The Application Scripts folder contains scripts that send instructions to the Corel application you are customizing. The General Scripts folder contains scripts that are not application specific.

   Based on a typical Corel installation, application scripts reside in the **C:\COREL*CorelSuite**application*\SCRIPTS** folder, where ***CorelSuite*** refers to the Corel products installed and ***application*** refers to the Corel application's folder. For example, CorelDRAW 7 scripts may reside in **C:\COREL\DRAW70\DRAW\SCRIPTS** folder and Corel VENTURA 7 scripts may reside in the **C:\COREL\VENTURA7\VENTURA\SCRIPTS** folder.

   General scripts normally reside in the **C:\COREL*CorelSuite*\SCRIPTS** folder.
4. Click the script.
5. In the Menu box, click the menu or sub-menu where you want to add the command.
6. Click Add.

Tip

- You can also assign a Corel SCRIPT Binaries to a menu by using the above procedure.
- Use the Separator button to add organizational lines to your menus.

{button ,AL(`cs_custom;;;;;',0,"Defaultoverview",)} Related Topics

To assign a Corel SCRIPT script to a toolbar button

From your Corel application or from the Corel SCRIPT Editor:

1. Activate the toolbar you want to edit.
2. Click Tools, Customize.
3. In the Commands box, double-click the Application Scripts folder or the General Scripts folder.

   The Application Scripts folder contains scripts that send instructions to the Corel application you are customizing. The General Scripts folder contains scripts that are not application specific.

   Based on a typical Corel installation, application scripts reside in the **C:\COREL*CorelSuite**application*\SCRIPTS** folder, where ***CorelSuite*** refers to the Corel products installed and ***application*** refers to the Corel application's folder. For example, CorelDRAW 7 scripts may reside in **C:\COREL\DRAW70\DRAW\SCRIPTS** folder and Corel VENTURA 7 scripts may reside in the **C:\COREL\VENTURA7\VENTURA\SCRIPTS** folder.

   General scripts normally reside in the **C:\COREL*CorelSuite*\SCRIPTS** folder.
4. Click the script.
5. Drag the appropriate command button to the toolbar. Right-click to cancel the movement.

Tip

- You can also assign a Corel SCRIPT Binaries to a toolbar button by using the above procedure.
- If a script's first line, second line, or both are REM statements, they are displayed in the Description text box.

{button ,AL(`cs_custom;;;;;',0,"Defaultoverview",)} Related Topics

To assign an shortcut key to a command

1. Click Tools, Customize.
2. Click Keyboard.
3. In the Commands box, double-click the command category folder containing the command you want to customize.
4. Click the command.
5. Click the Press new shortcut key box.
6. Press the keyboard combination that you want to assign to the command. If you need to make a correction, press the Backspace key.

   You can have up to four layers of keystrokes. For example, the key combination CTRL+ALT+1,2,3,4 is accomplished by holding down the CTRL and ALT keys, then pressing the 1,2,3, and 4 keys in succession.

Tip

- To automatically resolve shortcut conflicts, enable the Go to conflict on assign option.

{button ,AL(`cs_key_proc;;;;;',0,"Defaultoverview",)} Related Topics

To remove an shortcut key from a command

1. Click Tools, Customize.
2. Click Keyboard.
3. In the Commands box, double-click the command category folder containing the command you want to customize.
4. Click the command.
5. In the Current shortcut keys box, select the key combination you want to remove.
6. Click Delete.

{button ,AL(`cs_key_proc;;;;;',0,"Defaultoverview",)} Related Topics

To restore all keyboard assignments to their original settings

1. Click Tools, Customize.
2. Click Keyboard.
3. Click Reset All.

{button ,AL(`cs_key_proc;;;;;',0,"Defaultoverview",)} Related Topics

To save a set of customized keyboard assignments

1. Click Tools, Customize.
2. Click Keyboard.
3. Click Save As.
4. Choose the Shortcut File in which you want to save your assignments.

{button ,AL(`cs_key_proc;;;;;',0,"Defaultoverview",)} Related Topics

To load a set of customized keyboard assignments

1. Click Tools, Customize.
2. Click Keyboard.
3. Click Load.
4. Choose the Shortcut File you want to load.

{button ,AL(`cs_key_proc;;;;;',0,"Defaultoverview",)} **Related Topics**

To change the order of menus and menu commands

1. Click Tools, Customize.
2. Click Menu.
3. In the Menu box, click the menu or menu command you want to move. Double-click to open a menu or sub-menu.
4. Click Move Up or Move Down.

{button ,AL(`cs_menu_proc;;;;;',0,"Defaultoverview",)} **Related Topics**

To add a command to a menu

1. Click Tools, Customize.
2. Click Menu.
3. In the Commands box, double-click the command category folder containing the command you want to add.
4. Click the command.
5. In the Menu box, click the menu or sub-menu where you want to add the command.
6. Click Add.

Tip

- Use the Separator button to add organizational lines to your menus.

{button ,AL(`cs_menu_proc;;;;;',0,"Defaultoverview",)} Related Topics

To remove a menu or menu command

1. Click Tools, Customize.
2. Click Menu.
3. In the Menu box, click the menu or menu command you want to remove. Double-click to open a menu or sub-menu.
4. Click Remove.

{button ,AL(`cs_menu_proc;;;;;',0,"Defaultoverview",)} **Related Topics**

To rename a menu

1. Click Tools, Customize.
2. Click Menu.
3. In the Menu box, click the menu or menu command you want to rename. Double-click to open a menu or sub-menu.
4. Click the command's name tag, and type the new name.

{button ,AL(`cs_menu_proc;;;;;',0,"Defaultoverview",)} **Related Topics**

To change a menu command's shortcut key

1. Click Tools, Customize.
2. Click Menu.
3. In the Menu box, click the menu or menu command you want to rename. Double-click to open a menu or sub-menu.
4. Click the command's name tag, and insert an ampersand (&) before the letter you want to use as an shortcut.
5. Remove all unnecessary ampersands.

{button ,AL(`cs_menu_proc;cs_key_proc;;;;',0,"Defaultoverview",)} **Related Topics**

To add a new menu

1. Click Tools, Customize.
2. Click Menu.
3. Click Add Menu.
4. Type a name for the new menu.

Tip

- You can add a sub-menu to an existing menu by first double-clicking the existing menu.

{button ,AL(`cs_menu_proc;;;;;',0,"Defaultoverview",)} Related Topics

To restore the original menu settings

1. Click Tools, Customize.
2. Click Menu.
3. Click Reset All.

Note

- You will lose all changes to the menu settings.

{button ,AL(`cs_menu_proc;;;;;',0,"Defaultoverview",)} **Related Topics**

To move a toolbar

1. Click the border of the toolbar.
2. Drag it to its new location.

   Right-click to cancel the movement.

Tip

- Double-click a toolbar's title or border to automatically dock and undock it.

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} Related Topics

To resize a toolbar

1. Move the cursor to the edge of a floating toolbar.
2. Drag the edge until the toolbar is the correct size.
   Right-click to cancel the movement.

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} Related Topics

To activate and hide an existing toolbar

1. Click View, Toolbars.
2. Click the checkbox next to the toolbar you want to display or hide and click OK.

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

To create a custom toolbar

1. Click View, Toolbars.
2. Click New.
3. Type a name for the new toolbar.
4. Use the Customize command to add commands buttons to the new toolbar (See Related Topics).

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

To add a button to a toolbar

1. Activate the toolbar you want to edit (See Related Topics).
2. Click View, Toolbars.
3. Click Customize.
4. In the Commands box, click the command category folder containing the command you want to add.
5. Drag the appropriate command button to the toolbar. Right-click to cancel the movement.

Tip

- Click a button to see its description.
- You can also hold down the CTRL and ALT keys and drag a button to copy it to another toolbar without opening the dialog box.

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} Related Topics

To remove a button from a toolbar

1. Activate the toolbar you want to edit (See Related Topics).
2. Click View, Toolbars.
3. Click Customize.
4. Drag the button off the toolbar. Right-click to cancel the movement.

Tip

- You can also hold down the ALT key and drag a button off a toolbar to delete it without opening the dialog box.

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

To rename a toolbar

1. Click View, Toolbars.
2. Click the toolbar you want to rename.
3. Click the toolbar's name tag.
4. Type the new name.

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} Related Topics

To move a toolbar button

1. Activate the toolbar you want to edit (See Related Topics).
2. Click View, Toolbars.
3. Click Customize.
4. Drag the button to another toolbar, or to another spot on the same toolbar. Right-click to cancel the movement.

Tip

- To add space between two toolbar buttons, drag the right-most button slightly further to the right.
- You can also hold down the ALT key and drag a button to move it without opening the dialog box. To copy a button hold down the ALT and CTRL keys.

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

To delete a custom toolbar

1. Click View, Toolbars.
2. Click the custom toolbar you want to delete.
3. Click Delete.

Note

- You cannot delete a built-in toolbar.

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} Related Topics

To restore the original configuration of a built-in toolbar

1. Click View, Toolbars.
2. Click the built-in toolbar you want to reset.
3. Click Reset.

{button ,AL(`cs_toolbar_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

To view or hide status bar

- Click View, Status Bar.

  A check mark beside the Status Bar menu command indicates the status bar is displayed.

{button ,AL(`cs_status_bar;;;;;',0,"Defaultoverview",)} Related Topics

To move the status bar to the top or bottom of the Corel SCRIPT Editor

1. Right-click the status bar and select Position.
2. Click Top or Bottom.

Tip

- You can also move the status bar by double-clicking it.

{button ,AL(`cs_status_bar;;;;;',0,"Defaultoverview",)} **Related Topics**

To change what the status bar displays

1. Right-click the status bar and select Customize.
2. From the Command Categories list click Status Info.
3. Do the following:

   To add a field of a status information, drag the field from the Buttons area of the Customize dialog box over the Status bar

   To remove a field of status information, drag it from the status bar on to the document window.
4. Click OK.

{button ,AL(`cs_status_bar;;;;;',0,"Defaultoverview",)} **Related Topics**

To change the height of the status bar

1. Right-click the status bar and select Size.
2. Click Single Height or Double Height.

Tip

- You can also change the height of the status bar by dragging the top or bottom border.

{button ,AL(`cs_status_bar;;;;;',0,"Defaultoverview",)} **Related Topics**

To reset the status bar to its original configuration

- Right-click the status bar and select Reset Status Bar.

{button ,AL(`cs_status_bar;;;;;',0,"Defaultoverview",)} **Related Topics**

Tools, Options, Customize dialog box

Customize

Displays a list of the available toolbars. A check mark beside a toolbar indicates an active toolbar.

Use the Button slider to resize the toolbar buttons.

Use the Border slider to resize the button borders.

Enable to show titles on floating toolbars.

Enable to show text below bitmap images on toolbar buttons.

Creates a new custom toolbar.

Deletes a custom toolbar. You cannot delete a toolbar that comes with the application.

Displays the available commands. Double-click a command category to open it.

Shows the new keyboard combination that you want to assign to the command. If you need to make a correction, press the BACKSPACE key.

You can have up to four layers of keystrokes. For example, the key combination CTRL + ALT + 1, 2, 3, 4 is accomplished by holding down the CTRL and ALT keys, then pressing the 1, 2, 3, and 4 keys in succession.

Displays any commands assigned to the keyboard combination you typed. You cannot have the same combination for more than one command.

Enable to delete conflicting shortcut key.

Enable to highlight the command which no longer has a keyboard shortcut. You will then be prompted to enter a new shortcut in the Press New Shortcut Key box.

Displays any existing shortcut keys for the current command.

The name of the current keyboard assignment set.

Gives a short description of the selected shortcut.

Assigns the new keyboard combination to the current command.

Deletes the selected shortcut keys.

Resets the keyboard assignments to their original configuration.

Displays the commands, table assignment, key stroke combination and description associated with each shortcut key.

View All

Displays the list of available shortcut keys and their names. The current workspace is also displayed.

Opens the Save As dialog box which allows you to save your keyboard shortcuts as a text file.

Click this button to access the Keyboard Shortcuts dialog box, which lets you save your keyboard shortcuts as a text file or print them directly to your printer.

Closes this dialog without saving any attributes.

Adds the selected command to the menu.

Removes the selected command from the menu.

Adds a separating line to a menu below the current selection.

Adds a new menu.

Moves the current menu or menu entry up.

Moves the current menu or menu entry down.

Displays the current menu structure. Double-click a menu or submenu to open it.

Gives a short description of the selected command.

Resets the menu assignments to their original configuration.

Displays the command buttons for the current command category. Click a button to see its description, or drag it to add it to any toolbar on the screen.

Choose the Property Bar you want to customize from this list box. Drag the appropriate toolbar item icon to the Property Bar on your screen to add it to that Property Bar.

Gives a short description of the selected toolbar command.

Toggles between wide and narrow color swatch borders.

Toggles between large and small color swatches.

Shows and hides the No Color swatch.

Specifies the number of rows of colors to be displayed while the Color Palette is docked.

Changes the effect of right-clicking a color swatch on the palette.

Holding down the right mouse button for one second on the Color Palette, display a pop-up menu.

Displays the Roll-Ups and Roll-Up groups that arrange to the left side of the screen.

Displays the Roll-Ups and Roll-Up groups that arrange to the right side of the screen.

Moves the current Roll-Up or Roll-Up group from the right list to the left list.

Moves the current Roll-Up or Roll-Up group from the left list to the right list.

Adds a new, empty Roll-Up group to the left list.

Displays the name of the Roll-Up configuration that will appear on start up.

Allows you to change toolbar buttons so that text appears instead of bitmaps.

The text that appears in this box will now appear in the toolbar instead of the bitmap. Or, you can change the text to anything you like.

Allows you to change the bitmaps that appear in toolbar buttons. Use the controls shown to change the appearance of the bitmap.

Allows you to change the bitmap as displayed in the Preview window. Click one of the color swatches shown in the Color Palette, then click inside the Preview Window with the left mouse button.

Displays the four colors that are used in the creation of a typical button: dark gray (for shadows), white (for highlights), light gray (for the face), and black (for the text).

Click one of the color swatches shown, then click inside the Preview window with the left mouse button.

Click a color in either of the color palettes, then click inside the Preview window with the left mouse button in the grid to fill squares, or click with the right mouse button in the grid to erase squares.

Shows a preview of what the button will look like in its three states. The first example is how the button will appear on the toolbar when it is available. The second option shows how the button will appear when it is not available. And, the third option shows how the button will appear when it is depressed.

Click the Restore Defaults button to reverse all changes that you have made to the button.

Customize Keyboard

Changes the built-in keyboard assignments for accessing menu commands. You can create several sets of keyboard assignments to use for different types of projects, saving and loading sets as they are needed.

{button ,AL(`cs_custom_ov_cs;;;;;',0,"Defaultoverview",)} Related Topics

Customize Menus

Enables you to add commands to existing menus, or add new menus to the menu bar. You can change the order of the menus and their commands to give you quick, easy access to the functions you use most.

Note

- When you customize your menus, keep in mind that the help topics referring to those menus do not change.

{button ,AL(`cs_custom_ov_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

Customize Toolbars

Enables you to add and remove buttons and other controls on the toolbars as well as customize the information that appear in the Status Line. You can also create your own toolbars containing only the buttons and controls you use most often.

{button ,AL(`cs_custom_ov_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

About dialog box

Displays copyright information about this product.

Displays copyright information regarding this product.

Opens the Serial/PIN dialog box where you can enter your serial and personal identification number (PIN) for the product.

Displays liscencing information regarding this product.

Displays the registration information about this product.

Displays copyright and licensing information regarding this product.

Opens the System Info dialog box where you can get information about your system, display, printing, Corel EXE and DLL files, and system DLL files.

Displays the system information for the chosen category.

Saves the displayed system information as a text file (SYSINFO.TXT) in the X:\COREL\GRAPHICS8\CONFIG folder, where "X" indicates the drive where you installed the application.

Prints the copyright and licensing information regarding this product.

Displays the copyright and licensing information regarding this product.

Displays the personal identification number (PIN). This number is not needed to run the software but is necessary to receive customer support.

Displays the serial number located on your proof of purchase.

BUILDDATE function

BUILDDATE (Year, Month, Day)

Assigns a date value to a date variable.

Parameter	Description
Year	Numeric expression specifying the year to assign to a date variable.
Month	Numeric expression specifying the month to assign to a date variable. Valid values are from 1 to 12 inclusive.
Day	Numeric expression specifying the day to assign to a date variable. Valid values are from 1 to 31 inclusive, depending on the **Month** setting.

Note

- **BUILDATE** can only accept a date value between January 1, 1980 (date serial number 29221) and December 31, 2099 (date serial number 73050). If a date outside this range is specified, an error occurs.

Example

```
DIM BigDay AS DATE
BigDay = BUILDDATE(1996, 1, 14)
```

In the above example, the first line declares the date variable **BigDay**. This variable is then assigned the date January 14, 1996.

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} Related Topics

BUILDTIME function

BUILDTIME (Hour, Minute, Second)

Assigns a time value to a date variable.

Parameter	Description
Hour	Numeric expression specifying the hour to assign to a date variable. This value is based on a 24-hour clock. For example, 5 PM equals 17. Valid values are from 0 to 23 inclusive.
Minute	Numeric expression specifying the minute to assign to a date variable. Valid values are from 0 to 59 inclusive.
Second	Numeric expression specifying the second to assign to a date variable. Valid values are from 0 to 59 inclusive.

Example

```
DIM BigDay AS DATE
BigDay = BUILDTIME(14, 30, 0)
```

In the above example, the first line declares the date variable **BigDay**. This variable is then assigned the time 2:30 PM.

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} Related Topics

FORMATDATE function

DateString = FORMATDATE (DateExp, FormatString)

Converts a date expression into a string with a specified format.

Return Value

Lets you specify the string variable that is assigned the formatted date.

Parameter	Description
DateExp	Lets you specify the date expression to convert to a string.
FormatString	Lets you specify a code, as a string, representing the format of the date. Formats are created by using and combining the following codes.

To format	Use this format code (case-sensitive)
Days as 1-31	d
Days as 01-31	dd
Days as Sun-Sat	ddd
Days as Sunday-Saturday	dddd
Months as 1-12	M
Months as 01-12	MM
Months as Jan-Dec	MMM
Months as January-December	MMMM
Year as 6 in 1996	y
Year as 96 in 1996	yy
Year as 1996	yyy

Note

- You can insert spaces and punctuation between date elements within the formatting string. See the example below.

Example

```
DIM TodayDate AS DATE
TodayDate = GETCURRDATE()
StringDate$ = FORMATDATE (TodayDate, "dddd, MMMM d, yyy")
MESSAGE StringDate
```

In the above example, the first line declares the date variable **TodayDate**. This variable is then assigned the current date with the **GETCURRDATE** function. The **StringDate** variable is then assigned today's date using the formatting shown in the following example.

Saturday, September 16, 1995

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} **Related Topics**

FORMATTIME function

TimeString = FORMATTIME (TimeExp, FormatString)

Converts a time expression into a string with a specified format.

Return Value

Lets you specify the string variable that is assigned the formatted time.

Parameter	Description
TimeExp	Lets you specify the time expression to convert to a string.
FormatString	Lets you specify a code, as a string, representing the format of the time. Formats are created by using and combining the following codes.

To format	Use this format code (case-sensitive)
Hours as 1-12 (12-hour clock)	h
Hours as 01-12 (12-hour clock)	hh
Hours as 0-23 (24-hour clock)	H
Hours as 00-23 (24-hour clock)	HH
Minutes as 0-59	m
Minutes as 00-59	mm
Seconds as 0-59	s
Seconds as 00-59	ss
AM/PM as A or P	t
AM/PM as AM or PM	tt
Time as 4:36 pm	h:mm pm

Note

- You can insert spaces and punctuation between time elements within the formatting string. See the example below.

Example

```
DIM TimeNow AS DATE
TimeNow = GETCURRDATE()
StringTime = FORMATTIME (TimeNow, "HH:mm:ss tt")
MESSAGE StringTime
```

In the above example, the first line declares the date variable **TodayDate**. This variable is then assigned the current date and time with the **GETCURRDATE** function. The **StringTime** variable is then assigned the current time using the formatting shown in the following example.

```
13:46:25 PM
```

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} Related Topics

GETCURRDATE function

DateTime = GETCURRDATE ( )

Returns the system's current date and time.

Return Value

Lets you specify the date variable that is assigned the current date and time. See Assigning values to date variables for more information.

Note

- Formatting used to display dates and time is set in the Windows Control Panel. In Windows 95, see Regional settings for formatting information; in Windows NT, see International settings.
- **GETCURRDATE** can return a date between January 1, 1980 (date serial number 29221) and December 31, 2099 (date serial number 73050).
- In Corel SCRIPT version 7.0, the **GETCURRDATE** function and the **SETCURRDATE** statement replace the **CURRDATE** statement.

Example

```
DIM DT AS DATE
DT = GETCURRDATE()
MESSAGE DT
```

The above example displays the system date and time in a message box. You can also extract portions of the date and convert them to strings and numbers. The following continues the above example:

```
MyDateString$ = DT                ' create a string
MyDay% = VAL(LEFT(MyDateString$, 2)) ' extract the day as an integer
```

In the above example, the **VAL** and **LEFT** functions are used to extract an integer from the **MyDateString** string variable (created by converting a date variable). The method you use to extract portions from a date string depend on your Windows date settings. You can also use the **GETDATEINFO** or the **GETTIMEINFO** function to extract information from a date variable.

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} Related Topics

GETDATEINFO function

GETDATEINFO DateExp, Year, Month, Day, DayOfWeek

Extracts the components of a date expression to numeric variables.

Parameter	Description
DateExp	Lets you specify the date expression to extract components from.
Year	Lets you specify the numeric variable that is assigned the year component from the specified date expression.
Month	Lets you specify the numeric variable that is assigned the month component from the specified date expression.
Day	Lets you specify the numeric variable that is assigned the day component from the specified date expression.
DayOfWeek	Lets you specify the numeric variable that is assigned the day of week component from the specified date expression. Sunday corresponds to 1, Monday to 2, and so on.

Note

- **GETDATEINFO** can only accept a date value between January 1, 1980 (date serial number 29221) and December 31, 2099 (date serial number 73050). If a date outside this range is specified, an error occurs.

Example

```
DIM TodayDate AS DATE

TodayDate = GETCURRDATE()

GETDATEINFO TodayDate, Y&, M&, D&, DW&
```

In the above example, the first line declares the date variable **TodayDate**. This variable is then assigned the current date with the **GETCURRDATE** function. The variables **Y**, **M**, **D**, and **DW** are then assigned their respective component of the date stored in **TodayDate**. If **TodayDate** was set to May 29, 1996, then **Y**=1996, **M**=5, **D**=29, and **DW**=4.

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} **Related Topics**

GETTIMEINFO function

GETTIMEINFO TimeExp, Hour, Minute, Second

Extracts the components of a time expression to numeric variables.

Parameter	Description
TimeExp	Lets you specify the time expression to extract components from.
Hour	Lets you specify the numeric variable that is assigned the hour component from the specified time expression. The number assigned is based on a 24-hour clock. For example, 16 is the numeric variable for 4pm.
Minute	Lets you specify the numeric variable that is assigned the minute component from the specified time expression.
Second	Lets you specify the numeric variable that is assigned the second component from the specified time expression.

Example

```
DIM TodayTime AS DATE

TodayTime = GETCURRDATE()

GETTIMEINFO TodayTime, H&, M&, S&
```

In the above example, the first line declares the date variable **TodayTime**. This variable is then assigned the current date and time with the **GETCURRDATE** function. The variables **H**, **M**, and **S** are then assigned their respective component of the time stored in **TodayTime**. If **TodayTime** was set to 5:37:16 PM, then **H**=17, **M**=37, **S**=16.

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} **Related Topics**

SETCURRDATE statement

SETCURRDATE DT

Sets the system's date and time. If used improperly, this statement can cause problems in the system's Windows settings.

Parameter	Description
DT	Lets you specify a date expression that sets the system's date and time

Note

- **SETCURRDATE** can assign a date between January 1, 1980 (date serial number 29221) and December 31, 2099 (date serial number 73050).
- Formatting used to display dates and time is set in the Windows Control Panel. In Windows 95, see Regional settings for formatting information; in Windows NT, see International settings.
- In Corel SCRIPT version 7.0, the **SETCURRDATE** statement and the **GETCURRDATE** function replace the **CURRDATE** statement.

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} Related Topics

Examples for SETCURRDATE

To change the system date and time

The following example sets the system date to August 25, 1995, and the time to 12:00 A.M. A message box is used to display the date value of **dateOnly**.

```
DIM dateOnly AS DATE
dateOnly = 34936
SETCURRDATE dateOnly
MESSAGE dateOnly
```

The following example sets the system date to August 25, 1995, and the system time to 12:05.46 P.M. A message box is used to display the date value of **dateAndTime.**

```
DIM dateAndTime AS DATE
dateAndTime = 34936.504
CURRDATE = dateAndTime
MESSAGE dateAndTime
```

Using SETCURRDATE for file date stamping

The following example assigns the system date to the variable **xDate**. The third line sets the system date to February 26, 1994, and the fourth line opens a text file that is stamped with the new system date. The last line resets the system date to its original value.

```
DIM xDate AS DATE
xDate = GETCURRDATE      'assigns system date
SETCURRDATE "02/26/1994"     'sets the system date
OPEN "c:\log.txt" FOR OUTPUT AS 2
SETCURRDATE xDate
```

Note

- Formatting used to display dates and time is set in the Windows Control Panel. In Windows 95, see Regional settings for formatting information; in Windows NT, see International settings.

{button ,AL(`mid;left;val;cs_date_time;;',0,"Defaultoverview",)} Related Topics

WAIT FOR statement

WAIT FOR x

Pauses script execution for a specified number of seconds.

Argument	Description
x	A non-negative numeric expression specifying the number of seconds to pause script execution.

Example

MESSAGE "Start"

WAIT FOR 3

MESSAGE "Done"

The above example displays a message box, once the message box is closed, script execution pauses for three seconds, and then displays another message box.

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} Related Topics

WAIT UNTIL statement

WAIT UNTIL x

Pauses script execution until the system timer matches a specified date serial number.

You should consider using the **WAIT UNTIL** statement to run scripts in off-peak times. For example, if you have a large print job, you could use a script to run it during the night.

Argument	Description
x	Lets you specify the date and time when to resume execution. This parameter must be positive and greater than the system's current date.

Note

- **WAIT UNITL** can be set to a date between January 1, 1980 (date serial number 29221) and December 31, 2099 (date serial number 73050). If a date outside this range is specified, an error occurs.

Example

The following example pauses script execution until the system time matches 34936.25 (August 25, 1996, 6:00 A.M.).

```
DIM offPeak AS DATE
offPeak = 34936.25
WAIT UNTIL offPeak
```

The following example pauses script execution until 3:00 A.M. regardless of the date:

```
DIM today, tonite   AS DATE
DIM daypart AS LONG
today = CURRDATE
daypart = INT(today)   'daypart is set to 12:00 A.M. today
tonite = daypart + 1.125 'sets the date and time to 3 AM the next day
WAIT UNTIL tonite
```

If you add a day and 3 hours (1.125) to **datepart**, **tonite** is set to 3 A.M. the next morning.

{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} **Related Topics**

BEGIN DIALOG...END DIALOG statements (for static dialog boxes)

BEGIN DIALOG Identifier Left, Top, **Width, Height,** Text
    [dialog control statements]
END DIALOG

A user-defined static dialog box must begin with the **BEGIN DIALOG** statement and close with the **END DIALOG** statement.

The **BEGIN DIALOG** statement is followed by a series of statements that define a dialog. The series of statements insert dialog controls in the dialog box. Except for remarks statements, the only statements that can appear between **BEGIN DIALOG** and **END DIALOG** are the dialog control statements. The **END DIALOG** statement closes the definition of the user-defined dialog.

A user-defined static dialog can be changed by editing the statements between the **BEGIN DIALOG** and **END DIALOG** statements. An alternative to editing the statements is to use the dialog windows in the Corel SCRIPT Editor. Working in dialog windows is similar to using a drawing or painting application: dialog controls are graphic objects which can be inserted, moved, resized, and aligned in a dialog box.

The **BEGIN DIALOG** and **END DIALOG** statements on their own cannot display a dialog box and hold return values during a Corel SCRIPT script run. Use the **DIALOG** statement to display the dialog box.

Parameter	Description
Identifier	Name assigned to the dialog box sequence.
Left	Lets you specify the distance in dialog units from the dialog box's left border to the left side of the monitor's display area. If both Left and Top are omitted, the dialog box is centered on the screen.
Top	Lets you specify the distance in dialog units from the dialog box's top border to the top side of the monitor's display area. If both Left and Top are omitted, the dialog box is centered on the screen.
Width	Lets you specify the dialog box width in dialog units.
Height	Lets you specify the dialog box height in dialog units.
Text	Label displayed in the dialog box's title bar.
[dialog control statements]	Lets you specify the script statements that define the dialog's controls. These statements are called a dialog box's definition statements.

Note

- Dialog boxes are considered local to the procedure (main, or a user-defined subroutine or function) in which they are declared. However, if the dialog box uses global variables only, the dialog box can be called from any procedure in the script. Click for more information about global variables and variable availability.

{button ,AL(`corel_script_dialog_control;Returning_dialog_settings_and_choices;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`BEGIN_END_DIALOG_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

BITMAPBUTTON statement (for static dialog boxes)

BITMAPBUTTON Left, Top, Width, Height, Array

This statement adds a bitmap button to a dialog box. Pressing a bitmap button closes a dialog box and assigns the settings within the dialog box. Unlike push buttons, bitmap buttons display specified pictures instead of text. Bitmap buttons are often used in cases where another dialog box opens when the button is pressed.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the bitmap button in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the bitmap button in dialog units.
Width	Lets you specify the bitmap button width in dialog units.
Height	Lets you specify the bitmap button height in dialog units.
Array	Lets you specify a one-dimension string array containing a list of three Windows bitmaps (.BMP and .RLE files) and their paths. The button's state determines which bitmap in the array to reference and display: Array element 1 Normal state Array element 2 Depressed state Array element 3 Focus state If an array element entry is not available (for example, bad file name), the button is filled with black for the state corresponding to the unavailable element entry. If the array does not have an element for index number 1, 2, or 3, the button is filled with black for the state corresponding to the missing element index number, along with buttons that follow it in the array. For example, if array element 2 is missing, the bitmap button is filled with black for both the depressed and focus state. The array must be dimensioned before the dialog box definition in the script. The specified bitmaps are automatically scaled to fit within the button that has been specified.

Returns to dialog box	Condition
an integer from 3 to **n**	Pressing a bitmap button. Bitmap buttons and push buttons are numbered based on the order in which they are listed in a script, and not their placement within a dialog box. The first the push or bitmap button listed in a dialog box definition is set to 3. The second button is set to 4, and so on. The last push or bitmap button is equal to (2 + **n**) where **n** is the number of push and bitmap buttons. The values 1, and 2 are reserved for the OK button and Cancel button, respectively.

Note

- While editing a dialog box in the Corel SCRIPT Editor, a placeholder image is displayed in the dialog box.
- In addition to pressing a bitmap button, dialog boxes can also be closed by pressing a Cancel button, a push button, an OK button, or the Close Dialog button (). Dialog boxes are easier to use if they include an OK button and a Cancel button.
- Use the **ADDRESBMP** statement to embed bitmaps into an executable (.EXE) or DLL created with Corel SCRIPT, or a Corel SCRIPT Binary file (.CSB).

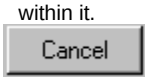
{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`BITMAPBUTTON_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

CANCELBUTTON statement (for static dialog boxes)

CANCELBUTTON Left, Top, Width, Height

This statement adds a Cancel button to a dialog box. Pressing a Cancel button closes a dialog box and discards the settings within it.



Parameter	Description
Left	Lets you specify the distance in dialog units from the inside of the dialog box's left border to the left side of the Cancel button.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the Cancel button in dialog units.
Width	Lets you specify the Cancel button width in dialog units.
Height	Lets you specify the Cancel button height in dialog units.

Returns to dialog box	Condition
2	Pressing the Cancel button to close the dialog box.

Note

- The Cancel button can't be assigned a shortcut key, but pressing the Close Dialog button ( ) is the same as pressing the Cancel button; both return 2 to the dialog box. Only one Cancel button in a dialog box is suggested.
- In addition to the Close Dialog button ( ), dialog boxes can also be closed by pressing an OK button, a Bitmap button, or a Push button.
- Dialog boxes are easier to use if they include an OK button and a Cancel button.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`CANCELBUTTON_DYN;;;;;',1,"Defaultoverview",)} **Syntax for dynamic dialog boxes**

CHECKBOX statement (for static dialog boxes)

CHECKBOX Left, Top, Width, Height, Text, Value

This statement adds a check box to a dialog box. A check box is used to present users with non-exclusive choices. To present users with exclusive choices, use option buttons.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the check box in dialog units.
Top	Lets you specify the distance in dialog units from the bottom of the dialog box's title bar to the top of the check box in dialog units.
Width	Lets you specify the check box and label width in dialog units.
Height	Lets you specify the check box height in dialog units.
Text	Lets you specify the label displayed to the right of the check box. Placing an ampersand (&) before a character provides a keyboard shortcut to select the check box.
Value	Lets you specify the variable that holds the return value that corresponds to the state of the check box. Optionally, you can use **Value** to set the default state of the check box.

Returns and Defaults	Condition
0	Check box is disabled, and empty.
1	Check box is enabled, and displays a check mark.
2	Grayed check box. A check box filled with gray indicates that a multiple selection contains a mix of property values. For example, selecting text that uses different fonts returns a mixed value.

Note

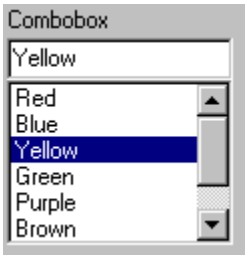
- Users can cycle through the three check box states by clicking the check box itself, or by clicking its label.
- To make a dialog box easier to read and understand, put a group box with a label around related check boxes.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`CHECKBOX_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

COMBOBOX statement (for static dialog boxes)

COMBOBOX Left, Top, Width, Height, Array, Value



This statement adds a combo box to a dialog box. As its name indicates, a combo box is a combination of boxes; in this case a combination of a text box and a list box. From a combo box, you can make a selection from a restricted set of items or enter your own selection in the text window. The advantage of a combo box over a list box is that you don't restrict a user to predefined items. A default selection can be provided in a combo box, and a string associated with a selection is returned to the script.

Each item in the list box portion of a combo box comes from a previously defined one-dimension array. The items should be listed in a logical order such as alphabetical, ascending sort order for number values, or some other logical sort order that is appropriate for the items in the list.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the combo box in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the combo box in dialog units.
Width	Lets you specify the combo box width in dialog units.
Height	Lets you specify the combo box height in dialog units.
Array	Lets you specify a one-dimension array containing a string list. The array must be dimensioned before the dialog box in the script.
Value	**Value** is a string variable that holds the return value that corresponds to the selected combo box entry or the user-entered text. Optionally, you can use **Value** to set the default selection in the combo box.

Returns and Defaults	Condition
a string	Returns a string that corresponds to the array element selected or the user-entered text.

Note

- If the items can't vertically fit into the list box portion, the list box automatically takes on a vertical scroll bar. The scroll bar is placed on the right-side of the list box to the immediate left of the right border. Having the scroll bar placed within the list box cuts into the item display area, so you should ensure that each list box entry can at least be recognized.
- If you type text in the text box, the list box scrolls to the closest matching item in the list box portion of the combo box. Choosing a list item from the list box replaces the text box contents with the list box selection.
- Because a combo box doesn't have a label component, a text control should be used to identify the combo box. The text control label can also be used to provide a shortcut to the combo box.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`COMBOBOX_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

DDCOMBOBOX statement (for static dialog boxes)

DDCOMBOBOX Left, Top, Width, Height, Array, Value

This statement adds a drop-down combo box to a dialog box. As its name indicates, a drop-down combo box is a combination drop-down list box and text box. The advantage of using a drop-down combo box over a regular combo box is that the control uses less space. In some cases, this might make the dialog easier to read. The drawback is that it adds another level of user interaction to a dialog box.

Each item in the list box portion of a drop-down combo box comes from a previously defined one-dimension array. The items should be listed in a logical order such as alphabetical, ascending sort for number values, or some other logical sort that is appropriate for the items in the list.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the drop-down combo box in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the drop-down combo box in dialog units.
Width	Lets you specify the drop-down combo box width in dialog units.
Height	Lets you specify the drop-down combo box height in dialog units when opened.
Array	Lets you specify a one-dimension array containing a string list. The array must be dimensioned before the dialog box in the script.
Value	**Value** is a string variable that holds the return value that corresponds to the selected drop-down combo box entry or the user-entered text. Optionally, you can use **Value** to set the default selection in the drop-down combo box.

Returns and Defaults	Condition
a string	Returns a string that corresponds to the array element selected or the user-entered text.

Note

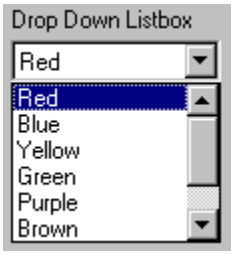
- If you type text in the text box, the list box scrolls to the closest matching item in the list box portion of the drop-down combo box. Choosing a list item from the list box replaces the text box contents with the list box selection.
- Because a drop-down combo box doesn't have a label component, a text control should be used to identify the dialog control. The text control label can also be used to provide a shortcut to the drop-down combo box. A default selection can be provided in a drop-down combo box.
- If the items can't vertically fit into the list box, the drop-down list box automatically takes on a vertical scroll bar. The scroll bar is placed on the right-side of the list box to the immediate left of the right border. Having the scroll bar placed within the list box cuts into the item display area, so you should ensure that each list box entry can at least be recognized.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`DDCOMBOBOX_DYN;;;;;',1,"Defaultoverview",)} **Syntax for dynamic dialog boxes**

DDLISTBOX statement (for static dialog boxes)

DDLISTBOX Left, Top, Width, Height, Array, Value



This statement adds a drop-down list box to a dialog box. A drop-down list box is like a list box, except to display the items and to make a choice, you must open the drop-down list box. Once a selection is made, the drop-down portion of the control closes.

The advantage of using a drop-down list box over a list box is that the control uses less space. In some cases, this might make the dialog easier to read. The drawback is that it adds another level of user interaction to a dialog box. Like the list box, the drop-down list box provides a user with a single choice from a restricted set of items. Each item in a drop-down list box comes from a previously defined one-dimension array. The items should be listed in a logical order such as alphabetical, ascending sort order for number values, or some other logical sort order that is appropriate for the items in the list.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the drop-down list box in dialog units.
Top	Lets you specify the distance in dialog units from the bottom of the dialog box's title bar to the top of the drop-down list box in dialog units.
Width	Lets you specify the drop-down list box width in dialog units.
Height	Lets you specify the drop-down list box height when opened in dialog units.
Array	Lets you specify a one-dimension array containing a string list. The array must be dimensioned before the dialog box in the script.
Value	**Value** is a variable that holds the return value that corresponds to the selected drop-down list box entry. Optionally, you can use **Value** to set the default selection in the drop-down list box.

Returns and Defaults	Condition
an integer	Returns an integer that corresponds to the array element selected.

Note

- If the items can't vertically fit into the list box, the drop-down list box automatically takes on a vertical scroll bar. The scroll bar is placed on the right-side of the list box to the immediate left of the right border. Having the scroll bar placed within the list box cuts into the item display area, so you should ensure that each list box entry can at least be recognized.
- Because a drop-down list box doesn't have a label component, a text control should be used to identify the drop-down list box. The text control label can also be used to provide a shortcut to the drop-down list box. A default selection can be provided in a drop-down list box and an integer associated with a selection is always returned to the script.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`DDLISTBOX_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

DIALOG statement (for static dialog boxes)

ReturnValue = DIALOG(Identifier)

This statement initializes, or displays a custom static dialog box, using a dialog box definition established earlier in the script. Use the function syntax to return a value corresponding to the button that was clicked to close the dialog box.

Parameter	Description
ReturnValue	The numeric variable assigned the return value corresponding to the button that was clicked to close the dialog box.
Identifier	Name assigned to the dialog box definition.

Returns	Condition
1	Pressing the OK button to close the dialog box.
2	Pressing the CANCEL button to close the dialog box. Pressing the Close Dialog button () is the same as pressing the Cancel button; both return 2.
an integer from 3 to **n**	Pressing a push or bitmap button. Push buttons and bitmap buttons are numbered based on the order in which they are listed in a script and not their placement within a dialog box. The first push or bitmap button listed in a dialog box definition is set to 3. The second button is set to 4, and so on. The last push or bitmap button is equal to (2 + **n**) where **n** is the number of push and bitmap buttons. The values 1, and 2 are reserved for the OK button and Cancel button, respectively.

Note

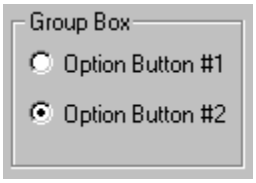
- Corel SCRIPT dialog boxes are modal; the running script cannot continue until the dialog box is closed.

{button ,AL(`corel_script_dialog_control;Returning_dialog_settings_and_choices;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`DIALOG_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

GROUPBOX statement (for static dialog boxes)

GROUPBOX Left, Top, Width, Height, Text



This statement adds a group box to a dialog box. A group box is a dialog control that doesn't return a value to a script, but is used to help arrange dialog controls to make the dialog box easier to understand. The group box is useful in physically grouping related check boxes and option buttons. The label component of the group box gives a user more information about a dialog.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the group box, in dialog units.
Top	Lets you specify the distance in dialog units from the bottom of the dialog box's title bar to the top of the group box, in dialog units.
Width	Lets you specify the group box width, in dialog units.
Height	Lets you specify the group box height, in dialog units.
Text	Lets you specify the label displayed at the top of the group box.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`GROUPBOX_DYN;;;;;',1,"Defaultoverview",)} **Syntax for dynamic dialog boxes**

HELPBUTTON statement (for static dialog boxes)

HELPBUTTON Left, Top, Width, Height, Text, Value

This statement adds a help button to a dialog box.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the help button, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the help button, in dialog units.
Width	Lets you specify the Help button width in dialog units.
Height	Lets you specify the Help button height in dialog units.
Text	A string or string variable that Lets you specify the help file (and it's path) that the Help button opens.
Value	Lets you specify the topic WinHelp displays whenever the help button is clicked. The topic is specified using a valid topic ID number. If a topic ID number is invalid and the help button is clicked, WinHelp opens an information dialog box notifying the user the topic is not available.

Note

- The help button control is an advanced control that should be used by Windows programmers and Windows help file authors. For more information about creating and compiling Windows help files and context-sensitive help, consult the Microsoft Windows SDK or the Microsoft Windows Help Author's Guide.
- Use a Windows Help Compiler to build a help file.
- The help button is not a push button and does not close a dialog box.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`HELPBUTTON_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

HSLIDER statement (for static dialog boxes)

HSLIDER Left, Top, Width, Height, Text, Value

This statement adds a horizontal slider control to a dialog box. The slider is used for setting and adjusting continuous numeric values. Some examples of appropriate uses for the slider include using it to adjust size, volume, or color intensity.

The slider indicator is used to adjust the control's return value. You can change the slider indicator position by dragging it horizontally with the mouse.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the slider in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the slider in dialog units.
Width	Lets you specify the slider width in dialog units.
Height	Lets you specify the slider height in dialog units.
Value	**Value** is the variable that holds the return value that corresponds to the slider indicator's position. Optionally, you can use **Value** to set the slider's indicator position.

Returns and Defaults	Condition
0 to 100 (an integer)	Corresponds to the position of the slider indicator. The left-most position is equal to 0, the right-most position is equal to 100, and the center position is equal to 50. A default value greater than 100 is set to 100, and a value less than 0 is set to 0.

Note

- Static slider controls display a tick mark at 0 and at 100.
- You can also move the slider indicator by clicking along the slider bar (20 units per click), or using the arrow keys (10 units per click) on the keyboard when the slider control has focus.
- A text control should be used to identify the slider, because a slider doesn't have a label component. The text control label can be used to provide a shortcut to the slider.
- You can use the **VSLIDER** statement to create a vertical slider.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`HSLIDER_DYN;;;;;',1,"Defaultoverview",)} **Syntax for dynamic dialog boxes**

IMAGE statement (for static dialog boxes)

IMAGE Left, Top, Width, Height, Value

This statement adds an image box to a dialog box. The image control can display Windows bitmaps (.BMP and .RLE files). The bitmap image is automatically sized to fit the height and width you've specified. As a dialog control, an image control doesn't return a value to a script, but can be used to provide or convey visual information to a user.

Parameter	Description
Left	Lets you specify the distance in dialog units from the inside of the dialog box's left border to the left side of the image box.
Top	Lets you specify the distance in dialog units from the bottom of the dialog box's title bar to the top of the image box.
Width	Lets you specify the Image box width in dialog units. The selected image is scaled to fit in the image box.
Height	Lets you specify the Image box height in dialog units. The selected image is scaled to fit in the image box.
Value	A string or string variable that Lets you specify the graphic (and its full path) to display in the image box.

Note

- You can insert Windows bitmaps (.BMP and .RLE files) into an image control.
- While editing a dialog box in Corel SCRIPT Editor, a placeholder image is displayed in the dialog box.
- Use the **ADDRESBMP** statement to embed bitmaps into an executable (.EXE) or DLL created with Corel SCRIPT, or a Corel SCRIPT Binary file (.CSB).

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`IMAGE_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

IMAGELISTBOX statement (for static dialog boxes)

IMAGELISTBOX Left, Top, Width, Height, Array, Value

Adds an image list box to a dialog box. The image list box is a dialog control used to preview and select Windows bitmaps (.BMP and .RLE files). You can select an image from the list box by clicking it. Like the list box, the image list box provides a user with a single choice from a restricted set of items. Each item in an image list box comes from a previously defined one-dimension array. Images displayed in an image list box are resized to fit horizontally.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the image list box in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the image list box in dialog units.
Width	Lets you specify the image list box width in dialog units.
Height	Lets you specify the image list box height in dialog units.
Array	Lets you specify a one-dimension array containing a string list of bitmap files (and their paths). The array must be dimensioned before the dialog box in the script. If a bitmap can't be found, it is filled with black in the image list box.
Value	**Value** is a variable that holds the return value that corresponds to the selected image list box entry. Optionally, you can use **Value** to set the default selection in the image list box.

Returns and Defaults	Condition
an integer	Returns an integer that corresponds to the array element selected.

Note

- Use the **ADDRESBMP** statement to embed bitmaps into an executable (.EXE) or DLL created with Corel SCRIPT, or a Corel SCRIPT Binary file (.CSB).

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`IMAGELISTBOX_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

LISTBOX statement (for static dialog boxes)

LISTBOX Left, Top, Width, Height, Array, Value

This statement adds a list box to a dialog box. A list box provides a user with a single choice from a restricted set of items and should be used instead of option buttons when there are more than six option items. Each item in a list box comes from a previously defined one-dimension array. The items should be listed in a logical order such as alphabetical, ascending sort for number values, or some other logical sort that is appropriate for the items in the list.

If the items can't vertically fit into the list box, the list box automatically takes on a vertical scroll bar. The scroll bar is placed on the right-side of the list box to the immediate left of the right border. Having the scroll bar placed within the list box cuts into the item display area, so you should ensure that each list box entry can be recognized.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the list box, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the list box, in dialog units.
Width	Lets you specify the list box width, in dialog units.
Height	Lets you specify the list box height, in dialog units.
Array	Lets you specify the one-dimension array containing a string list. The array must be dimensioned before the dialog box in the script.
Value	**Value** is a variable that holds the return value that corresponds to the selected list box entry. Optionally, you can use **Value** to set the default selection in the list box.

Returns and Defaults	Condition
an integer	Returns an integer that corresponds to the array element selected.

Note

- Because a list box doesn't have a label component, a text control should be used to identify the list box. The text control label can also be used to provide a shortcut to the list box. A default selection can be provided in a list box, and an integer associated with a selection is always returned to the script.

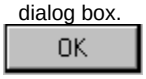
{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`LISTBOX_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

OKBUTTON statement (for static dialog boxes)

OKBUTTON Left, Top, Width, Height

This statement adds an OK button to a dialog box. Pressing an OK button closes a dialog box and assigns the settings within the dialog box.



Parameter	Description
Left	Distance in dialog units from the inside of the dialog box's left border to the left side of the OK button.
Top	Distance in dialog units from the bottom of the dialog box's title bar to the top of the OK button.
Width	OK button width in dialog units.
Height	OK button height in dialog units.

Returns to dialog box	Condition
1	Press the OK button to close the dialog box.

Note

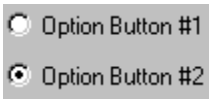
- The OK button can't be assigned a shortcut key, but pressing ENTER is the same as pressing the OK button (unless a push button or the Cancel button has focus). Only one OK button in a dialog box is suggested.
- In addition to pressing an OK button or pressing ENTER, dialog boxes can also be closed by pressing a Cancel button, a Bitmap button, a Push button, or the Close Dialog button ( ).
- Dialog boxes are easier to use if they include an OK button and a Cancel button.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`OKBUTTON_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

OPTIONBUTTON statement (for static dialog boxes)

OPTIONBUTTON Left, Top, Width, Height, Text



Adds an option button to a dialog box. Option buttons are used to present two or more mutually exclusive choices. Only one option button in a group can be selected, and selecting an option button in a group de-selects a previously selected button. One option button from a group always remains selected.

In a script, **OPTIONBUTTON** statements are grouped together without intervening statements (remarks excluded) and must be immediately preceded by an **OPTIONGROUP** statement. The selected option button is returned to the **OPTIONGROUP** variable **Value**.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the option button in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the option button in dialog units.
Width	Lets you specify the option button and label width in dialog units.
Height	Lets you specify the option button and label height in dialog units.
Text	Lets you specify the label displayed to the right of the option button. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting the option button.

Note

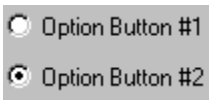
- Option buttons should be used to choose a property, or to set a value or an option. Consider using some type of list box when you have more than 6 choices.
- In dialog windows in the Corel SCRIPT Editor, option buttons are grouped as you create them. As you insert option buttons, they will automatically be included in the same group until you enter another type of control. Once you enter another type of control, and then insert more option buttons, the options buttons will form another group with an **OPTIONGROUP** statement preceding the first option button.
- To make a dialog box easier to read, put a group box around the option buttons and a label indicating the purpose of the option buttons.
- Option buttons are often called radio buttons, and are often confused with check boxes. Although they perform similar functions, option buttons allow only one option in a group of options while check boxes allow you multiple options.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`OPTIONBUTTON_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

OPTIONGROUP statement (for static dialog boxes)

OPTIONGROUP Value



This statement marks the beginning of a series of **OPTIONBUTTON** statements in a script. The option button statements are positioned directly below the OPTIONGROUP statement without any intervening statements (remarks excluded).

Parameter	Description
Value	Lets you specify a numeric variable that is assigned the return value that corresponds to the selected option button within the group. It can also be used to set the default enabled button.

Returns and defaults	Condition
an integer from 0 to n	The integer that corresponds to the option button selected by the user. The first option button in an option group is identified as 0, the second option button is identified as 1, and so on. The last option button is identified as **n**. The order of the option buttons is determined, not by their placement within the dialog box, but by the order in which they are listed in the script.

Note

- The Corel SCRIPT Editor groups option buttons as you create them in dialog windows. As you insert option buttons, they will automatically be included in the same group until you enter another type of control. Once you enter another type of control, and then insert more option buttons, the options buttons will form another group with an OPTIONGROUP statement preceding the first option button.

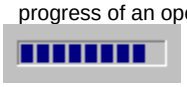
{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`OPTIONGROUP_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

PROGRESS statement (for static dialog boxes)

PROGRESS Left, Top, Width, Height, Value

This statement adds a progress indicator to a dialog box. In a dynamic dialog box, the progress indicator visually displays the progress of an operation. A progress indicator in a static dialog box cannot be used to provide the user with any information.



Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the progress indicator in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the progress indicator in dialog units.
Width	Lets you specify the progress indicator width in dialog units.
Height	Lets you specify the progress indicator height in dialog units.
Value	Lets you specify a default complete value for the progress indicator as a percentage.

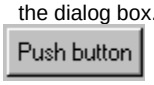
{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`PROGRESS_DYN;;;;;',1,"Defaultoverview",)} **Syntax for dynamic dialog boxes**

PUSHBUTTON statement (for static dialog boxes)

PUSHBUTTON Left, Top, Width, Height, Text

This statement adds a push button to a dialog box. Pressing a push button closes a dialog box and assigns the settings within the dialog box.



Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the push button in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the push button in dialog units.
Width	Lets you specify the push button width in dialog units.
Height	Lets you specify the push button height in dialog units.
Text	Label displayed on the push button. Placing an ampersand (&) before a character provides a keyboard shortcut to select a push button.

Returns to dialog box	Condition
an integer from 3 to **n**	Pressing a push button. Push buttons and bitmap buttons are numbered based on the order in which they are listed in a script, and not their placement within a dialog box. The first push or bitmap button listed in a dialog box definition is set to 3. The second button is set to 4, and so on. The last push or bitmap button is equal to (2 + **n**) where **n** is the number of push and bitmap buttons. The values 1, and 2 are reserved for the OK button and Cancel button, respectively.

Note

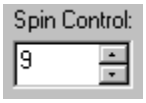
- Push buttons are often used to open another dialog box.
- In addition to pressing a push button, dialog boxes can also be closed by pressing a Cancel button, a bitmap button, an OK button, or the Close Dialog button ( ).
- Dialog boxes are easier to use if they include an OK button and a Cancel button.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`PUSHBUTTON_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

SPINCONTROL statement (for static dialog boxes)

SPINCONTROL Left, Top, Width, Height, Value



This statement adds a spin control to a dialog box. The spin control is used to change values in numeric entry text boxes by using the mouse or keyboard. The top arrow increases the value displayed, the bottom arrow decreases it. You can either click the arrow to change the value by a single increment, or hold the mouse button down on an arrow to cause the value to change continuously. You can also type directly into the text window portion.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the spin control in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the spin control in dialog units.
Width	Lets you specify the spin control width in dialog units.
Height	Lets you specify the spin control height in dialog units.
Value	**Value** is a variable that holds the return value that corresponds to the number in the spin control. Optionally, you can use **Value** to set the default value of the spin control. **Value** returns an integer that ranges from -32,768 to 32,767.

Returns and Defaults	Condition
a numeric variable	Returns a number that corresponds to the value selected, or entered, into the spin control.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} **Related Topics**

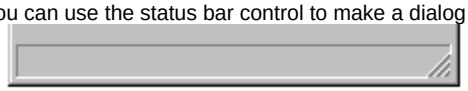
{button ,AL(`SPINCONTROL_DYN;;;;;',1,"Defaultoverview",)} **Syntax for dynamic dialog boxes**

STATUS statement (for static dialog boxes)

STATUS Text

This statement adds a status bar to a dialog box. The status bar is a window at the bottom of a dialog box that displays specified status text. Unlike most dialog controls, the status control does not have size or position properties, it is always placed at the bottom of a dialog box. As the dialog box is resized in the Corel SCRIPT Editor, the status bar control is also automatically resized.

You can use the status bar control to make a dialog box easier to understand; it does not return a value to a script.



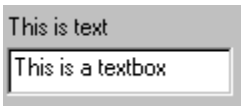
Parameter	Description
Text	Lets you specify the text to display in the status bar, beginning in the left corner.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`STATUS_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

TEXT statement (for static dialog boxes)

TEXT Left, Top, Width, Height, Text



This statement adds a text label to a dialog box. Text controls in dialog boxes are used as labels and to provide user instructions. As labels, text controls do not return a value back to a running script.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the text in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the text in dialog units.
Width	Lets you specify the text label bounding box width in dialog units.
Height	Lets you specify the text label bounding box height in dialog units.
Text	Lets you specify the text label to display in dialog box.

Note

- Use sentence formatting for dialog text titles; the initial character of the sentence in uppercase. The sentence should end with a colon, not a period. For example, **Choose a file:** You should provide text labels for dialog controls that don't have a label component.
- Controls that use labels such as push buttons and option buttons can have shortcut keys assigned to them by placing an ampersand (&) before a label character. When the script displays the dialog box, you can use a keyboard shortcut to select a control by pressing ALT and the underlined shortcut key (the character that follows the ampersand). You can also use the text control to provide shortcut keys for controls that don't use labels.
- A text control must be associated with an unlabelled dialog control (available controls listed below) to provide a keyboard shortcut. To associate a text control with a dialog control, the text control statement in the Corel SCRIPT script must immediately precede the unlabelled control statement. Association is not based on a control's dialog location.
- The following controls can be associated with a text control:

  text box

  list box

  drop-down list box

  combo box

  drop-down combo box

  spin control

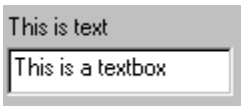
  image list box
- See Changing focus in dialog boxes for more information about using shortcut keys in a custom dialog box.
- You can insert a line break in text controls by using the CHR(13) function (as shown in the example). See the CHR function for more information about using special characters.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`TEXT_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

TEXTBOX statement (for static dialog boxes)

TEXTBOX Left, Top, Width, Height, Text



This statement adds a text box to a dynamic dialog box. The text box control receives user-inputted text which is returned to the script as a string.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the text box in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the text box in dialog units.
Width	Lets you specify the text box width in dialog units.
Height	Lets you specify the text box height in dialog units.
Text	Lets you specify a string variable that is assigned the text the user enters in the text box. This variable can also be used to set the default text in the text box.

Returns and Defaults	Condition
a string	Corresponds to the text the user enters into the text box or the default text.

Note

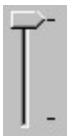
- If you want a number returned to a script from a text box, use the **VAL** function to convert the text because a text box can only return strings.
- You can use standard window shortcut keys such as CTRL + X to cut text, CTRL + C to copy text, and CTRL + V to paste text. Arrow keys can also be used in a text box to move the cursor.
- Because a text box doesn't have a label component, a text control should be used to identify the text box control. The text control label can be used to provide a shortcut to the text box.
- Pressing CTRL + ENTER in a text box inserts a line return character. Pressing ENTER or SHIFT + ENTER in the text box control is the same as pressing a push button, and closes the dialog box.
- You can scroll through the text horizontally using the arrow keys.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`TEXTBOX_DYN;;;;;',1,"Defaultoverview",)} Syntax for dynamic dialog boxes

VSLIDER statement (for static dialog boxes)

VSLIDER Left, Top, Width, Height, Value



This statement adds a vertical slider control to a dialog box. The slider is used for setting and adjusting continuous numeric values. Some examples of appropriate uses for the slider include using it to adjust size, volume, or color intensity.

The slider indicator is used to adjust the control's return value. You can change the indicator position by dragging it vertically with the mouse.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the slider in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the slider in dialog units.
Width	Lets you specify the slider width in dialog units.
Height	Lets you specify the slider height in dialog units.
Value	**Value** is a variable that holds the return value that corresponds to the slider indicator's position. Optionally, you can use **Value** to set the slider's indicator position.

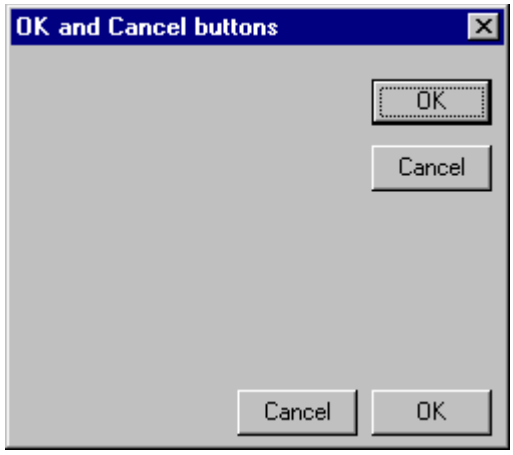
Returns and Defaults	Condition
0 to 100 (an integer)	Corresponds to the position of the slider indicator. The left-most position is equal to 0, the right-most position is equal to 100, and the center position is equal to 50. A default value greater than 100 is set to 100, and a value less than 0 is set to 0.

Note

- Static slider controls displays a tick mark at 0 and at 100.
- You can also move the slider indicator by clicking along the slider bar (20 units per click), or by using the arrow keys (10 units per click) on the keyboard when the slider control has focus.
- Because a slider doesn't have a label component, a text control should be used to identify the slider. The text control label can be used to provide a shortcut to the slider.
- You can use the **HSLIDER** statement to create a horizontal slider.

{button ,AL(`corel_script_dialog_control;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`VSLIDER_DYN;;;;;',1,"Defaultoverview",)} **Syntax for dynamic dialog boxes**

- 
OK and Cancel buttons
OK
Cancel
Cancel
OK

Dialog box definition statements

A dialog box definition consists of the BEGIN DIALOG and END DIALOG statements with control statements in between. The following list contains valid dialog box definition statements:

- BEGIN DIALOG
- BITMAPBUTTON
- CANCELBUTTON
- CHECKBOX
- COMBOBOX
- DDCOMBOBOX
- DDLISTBOX
- END DIALOG
- GROUPBOX
- HELPBUTTON
- HSLIDER
- IMAGE
- IMAGELISTBOX
- LISTBOX
- OKBUTTON
- OPTIONBUTTON
- OPTIONGROUP
- PROGRESS
- PUSHBUTTON
- SPINCONTROL
- STATUS
- TEXT
- TEXTBOX
- VSLIDER

You can include remark (REM) statements in a dialog box definition.

No related topics were found.

No procedure topics were found.

Specifies the name of the Dialog Event Handler subroutine (**Example1**) which corresponds to the subroutine specified in a **BEGIN DIALOG** statement.. The two required parameters are both passed by value (**BYVAL**). **ControlID** is the variable that is passed a value corresponding to the dialog control that is generating a dialog event. Event is the variable that is passed a value corresponding to the dialog event that occurs in the dialog box.

Conditional statements to execute when the dynamic dialog box is initialized, or first displayed. Typical uses for the Dialog Initialization event include setting default text in text boxes, specifying arrays for list boxes, and check box and spin control attributes.

Conditional statements to execute when a control is clicked in the dynamic dialog box. Clicking a control is the most common type of event in a dynamic dialog box and generally occurs whenever a push button is clicked or a selection is made in a check box, option button, or any type of list box.

Conditional statements to execute when the checkbox (the first control in the dialog box definition) is clicked.

Conditional statements to execute when a control is clicked in the dynamic dialog box. The nested conditional statements are executed if the check box (Control = 1) or option button (Control = 3) have been clicked.

Specifies the name of the Dialog Event Handler subroutine (**Example1**). This subroutine sends script instructions to a dynamic dialog box when an event or changes occur in a dialog box. For example, clicking a checkbox is a dialog event that can be used to trigger a change in a dialog box such as enabling or hiding a control.

Specifies a string variable that identifies the dynamic dialog control.
The period (.) before the identifier is required.

Specifies a check box setting in the dialog box. Any dynamic dialog functions that are executed before the **DIALOG** statement (which display the dialog box) set a dynamic dialog box's initial settings. However, it is better programming practice to place initialization statements in Dialog Event Handler subroutine in a Event = 0 condition.

BEGIN DIALOG...END DIALOG statements (for dynamic dialog boxes)

BEGIN DIALOG OBJECT Identifier Left, Top, **Width, Height,** Text, SUB Subname
    [dialog control statements]
END DIALOG

A user-defined dynamic dialog box must begin with the **BEGIN DIALOG** statement and close with the **END DIALOG** statement.

The **BEGIN DIALOG** statement is followed by a series of statements that define a dialog. The series of statements insert dialog controls into the dialog box. Except for remarks statements, the only statements that can appear between **BEGIN DIALOG** and **END DIALOG** are the dialog control statements. The **END DIALOG** statement closes the definition of the user-defined dialog.

A user-defined dynamic dialog can be changed by editing the statements between the **BEGIN DIALOG** and **END DIALOG** statements. An alternative to editing the statements is to use the dialog windows in the Corel SCRIPT Editor. Working in dialog windows is similar to using a drawing or painting application: dialog controls are graphic objects which can be inserted, moved, re-sized, and aligned in a dialog box.

The **BEGIN DIALOG** and **END DIALOG** statements on their own cannot display a dialog box and hold return values during a Corel SCRIPT script run. Use the **DIALOG** statement to display the dialog box.

Parameter	Description
OBJECT	Required script keyword.
Identifier	Name assigned to the dialog box sequence.
Left	Lets you specify the distance in dialog units from the dialog box's left border to the left side of the monitor's display area. If both Left and Top are omitted, the dialog box will be centered on the screen.
Top	Lets you specify the distance in dialog units from the dialog box's top border to the top side of the monitor's display area. If both Left and Top are omitted, the dialog box will be centered on the screen.
Width	Lets you specify the dialog box width in dialog units.
Height	Lets you specify the dialog box height in dialog units.
Text	Lets you specify a string or a string variable that is used to set the text in the dialog box's title bar.
SUB	Script keyword that precedes the **Subname** parameter, if used.
Subname	Lets you specify the name of Dialog Event Handler subroutine. This subroutine sends instruction to a dialog box based on events which occur in the defined dialog box.
[dialog control statements]	Lets you specify the script statements that define the dialog's controls. These statements are called a dialog box's definition statements.

Note

- You can use the **SETSTYLE** function to specify whether to display an icon and buttons in the dialog box's title bar.
- Dialog boxes are considered local to the procedure (main, or a user-defined subroutine or function) in which they are declared. However, if the dialog box uses global variables only, the dialog box can be called from any procedure in the script. Click for more information about global variables and variable availability.

{button ,AL(`corel_script_dialog_control_dynamic_dynamic;Returning_dialog_settings_and_choices;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`closedialog;move;getheight;getleftposition;gettopposition;getwidth;settext;gettext;setTIMER;getTIMER;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`BEGIN_END_DIALOG;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

BITMAPBUTTON statement (for dynamic dialog boxes)

BITMAPBUTTON Left, Top, Width, Height, .Identifier

This statement add a bitmap button to a dynamic dialog box. Unlike the static version of this control, this statement does not close a dialog box when clicked. Bitmap buttons are often used to open another dialog box. For more information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the bitmap button, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the bitmap button, in dialog units.
Width	Lets you specify the bitmap button width, in dialog units.
Height	Lets you specify the bitmap button height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

Note

- You can use the **SETSTYLE** function to change how bitmaps appear in a bitmap button.
- Use the **ADDRESBMP** statement to embed bitmaps into an executable (.EXE) or DLL created with Corel SCRIPT, or a Corel SCRIPT Binary file (.CSB).

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;setarray;enable;;;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`BITMAPBUTTON;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

CANCELBUTTON statement (for dynamic dialog boxes)

CANCELBUTTON Left, Top, Width, Height, .Identifier

This statement adds a Cancel button to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance in dialog units from the inside of the dialog box's left border to the left side of the Cancel button.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the Cancel button, in dialog units.
Width	Lets you specify the Cancel button width, in dialog units.
Height	Lets you specify the Cancel button height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`settext;gettext;enable;all_dyn_functions;;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`CANCELBUTTON;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

CHECKBOX statement (for dynamic dialog boxes)

CHECKBOX Left, Top, Width, Height, .Identifier, Text

This statement adds a check box to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the check box, in dialog units.
Top	Lets you specify the distance in dialog units from the bottom of the dialog box's title bar to the top of the check box, in dialog units.
Width	Lets you specify the check box and label width, in dialog units.
Height	Lets you specify the check box height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.
Text	Lets you specify a string or a string variable that is used to set the check box's initial text attribute. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting the check box.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;setvalue;getvalue;setthreestate;enable;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`CHECKBOX;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

COMBOBOX statement (for dynamic dialog boxes)

COMBOBOX Left, Top, Width, Height, .Identifier

This statement adds a combo box to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the combo box, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the combo box, in dialog units.
Width	Lets you specify the combo box width, in dialog units.
Height	Lets you specify the combo box height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;lb_funs;setselect;getselect;setarray;enable;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`COMBOBOX;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

DDCOMBOBOX statement (for dynamic dialog boxes)

DDCOMBOBOX Left, Top, Width, Height, .Identifier

This statement adds a drop-down combo box to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the drop-down combo box, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the drop-down combo box, in dialog units.
Width	Lets you specify the drop-down combo box width, in dialog units.
Height	Lets you specify the drop-down combo box height, in dialog units when opened.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`all_dyn_functions;lb_funs;setselect;getselect;setarray;enable;',0,"Defaultoverview",)} **Dynamic dialog functions**

{button ,AL(`DDCOMBOBOX;;;;;',1,"Defaultoverview",)} **Syntax for static dialog boxes**

DDLISTBOX statement (for dynamic dialog boxes)

DDLISTBOX Left, Top, Width, Height, .Identifier

This statement adds a drop-down list box to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Specified the distance from the inside of the dialog box's left border to the left side of the drop-down list box, in dialog units.
Top	Specified the distance in dialog units from the bottom of the dialog box's title bar to the top of the drop-down list box, in dialog units.
Width	Specified the drop-down list box width, in dialog units.
Height	Specified the drop-down list box height, in dialog units, when opened.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;lb_funs;setselect;getselect;setarray;enable;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`DDLISTBOX;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

DIALOG statement (for dynamic dialog boxes)

DIALOG Identifier

ReturnValue = DIALOG(Identifier)

This statement initializes, or displays a custom dynamic dialog box, using a dialog box definition established earlier in the script. Use the function syntax to return a value corresponding to the button that was clicked to close the dialog box.

Parameter	Description
ReturnValue	The numeric variable that is assigned the return value corresponding to the button that was clicked to close the dialog box.
Identifier	The name assigned to the dialog box definition in the **BEGIN DIALOG** statement.

Returns	Condition
1	Pressing the OK button to close the dialog box.
2	Pressing the CANCEL button to close the dialog box. Pressing the Close Dialog button () is the same as pressing the Cancel button; both return 2.
an integer >= 3	Triggering an event that uses the **CLOSEDIALOG** function.

Note

- Since push buttons and bitmap buttons can't be used to close a dynamic dialog box, they do not return a numeric value to the DIALOG function.

{button ,AL(`corel_script_dialog_control_dynamic;Returning_dialog_settings_and_choices;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`DIALOG;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

GROUPBOX statement (for dynamic dialog boxes)

GROUPBOX Left, Top, Width, Height, .Identifier, Text

This statement adds a group box to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the group box, in dialog units.
Top	Lets you specify the distance in dialog units from the bottom of the dialog box's title bar to the top of the group box, in dialog units.
Width	Lets you specify the group box width, in dialog units.
Height	Lets you specify the group box height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.
Text	Lets you specify a string or a string variable that is used to set the group box's initial text attribute.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`settext;gettext;enable;all_dyn_functions;;',0,"Defaultoverview",)} **Dynamic dialog functions**

{button ,AL(`GROUPBOX;;;;;',1,"Defaultoverview",)} **Syntax for static dialog boxes**

HELPBUTTON statement (for dynamic dialog boxes)

HELPBUTTON Left, Top, Width, Height, .Identifier

This command adds a help button to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the help button, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the help button, in dialog units.
Width	Lets you specify the Help button width, in dialog units.
Height	Lets you specify the Help button height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;sethelppath;gethelppath;sethelpindex;gethelpindex;enable;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`HELPBUTTON;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

HSLIDER statement (for dynamic dialog boxes)

HSLIDER Left, Top, Width, Height, .Identifier

This statement adds a horizontal slider control to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the slider, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the slider, in dialog units.
Width	Lets you specify the slider width, in dialog units.
Height	Lets you specify the slider height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`settick;gettick;setincrement;getincrement;setvalue;getvalue;setmaxrange;getmaxrange;setminrange;getminrange;enable;all_dyn_functions;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`HSLIDER;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

IMAGE statement (for dynamic dialog boxes)

IMAGE Left, Top, Width, Height, .Identifier

This command adds an image box to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance in dialog units from the inside of the dialog box's left border to the left side of the image box.
Top	Lets you specify the distance in dialog units from the bottom of the dialog box's title bar to the top of the image box.
Width	Lets you specify the Image box width, in dialog units. Your selected image is scaled to fit in the image box.
Height	Lets you specify the Image box height, in dialog units. Your selected image is scaled to fit in the image box.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

Note

- You can use the **SETSTYLE** function to change the properties of an image box. You can make the image box appear sunken or specify how a bitmap appears in an image control.
- You can use the **SETBITMAPOFFSET** function to specify a portion of a bitmap to display in the image box. Click for an example.
- Use the **ADDRESBMP** statement to embed bitmaps into an executable (.EXE) or DLL created with Corel SCRIPT, or a Corel SCRIPT Binary file (.CSB).

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;setimage;getimage;;;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`IMAGE;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

IMAGELISTBOX statement (for dynamic dialog boxes)

IMAGELISTBOX Left, Top, Width, Height, .Identifier

Adds an image list box to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the image list box, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the image list box, in dialog units.
Width	Lets you specify the image list box width, in dialog units.
Height	Lets you specify the image list box height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

Note

- Use the **ADDRESBMP** statement to embed bitmaps into an executable (.EXE) or DLL created with Corel SCRIPT, or a Corel SCRIPT Binary file (.CSB).

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;lb_funs;setselect;getselect;setarray;enable;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`IMAGELISTBOX;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

LISTBOX statement (for dynamic dialog boxes)

LISTBOX Left, Top, Width, Height, .Identifier

This statement adds a list box to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the list box, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the list box, in dialog units.
Width	Lets you specify the list box width, in dialog units.
Height	Lets you specify the list box height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;lb_funs;setselect;getselect;setarray;enable;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`LISTBOX;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

OKBUTTON statement (for dynamic dialog boxes)

OKBUTTON Left, Top, Width, Height, .Identifier

This statement adds an OK button to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Distance in dialog units from the inside of the dialog box's left border to the left side of the OK button.
Top	Distance in dialog units from the bottom of the dialog box's title bar to the top of the OK button.
Width	OK button width, in dialog units.
Height	OK button height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`settext;gettext;enable;all_dyn_functions;;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`OKBUTTON;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

OPTIONBUTTON statement (for dynamic dialog boxes)

OPTIONBUTTON Left, Top, Width, Height, .Identifier, Text

Adds an option button to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the option button, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the option button, in dialog units.
Width	Lets you specify the option button and label width, in dialog units.
Height	Lets you specify the option button and label height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.
Text	Lets you specify a string or a string variable that is used to set the option button's label initial text attribute. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting the option button.

Note

- Unlike in static dialog boxes, you can also use the **SETVALUE** and **GETVALUE** functions to set and return option button values.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`enable;all_dyn_functions;settext;gettext;setvalue;getvalue;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`OPTIONBUTTON;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

OPTIONGROUP statement (for dynamic dialog boxes)

OPTIONGROUP .Identifier

This statement marks the beginning of a series of **OPTIONBUTTON** statements in a script. The option button statements are positioned directly below the **OPTIONGROUP** statement, without any intervening statements (remarks excluded).

Parameter	Description
Identifier	Lets you specify a string variable that identifies the dynamic dialog box construct. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`setvalue;getvalue;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`OPTIONGROUP;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

PROGRESS statement (for dynamic dialog boxes)

PROGRESS Left, Top, Width, Height, .Identifier

This statement adds a progress indicator to a dialog box. In a dynamic dialog box, the progress indicator visually displays the progress of an operation by filling a control gauge from left-to-right, approximating the relative progress of a long script operation. The progress indicator doesn't return a value to a script and is only used to provide information to a user.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the progress indicator, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the progress indicator, in dialog units.
Width	Lets you specify the progress indicator width, in dialog units.
Height	Lets you specify the progress indicator height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`all_dyn_functions;setincrement;getincrement;setvalue;step;setmaxrange;getmaxrange;setminrange;getminrange;',0,"Defaultoverview",)} **Dynamic dialog functions**

{button ,AL(`PROGRESS;;;;;',1,"Defaultoverview",)} **Syntax for static dialog boxes**

PUSHBUTTON statement (for dynamic dialog boxes)

PUSHBUTTON Left, Top, Width, Height, .Identifier, Text

This statement adds push button to a dynamic dialog box. Unlike the static version of this control, it does not close a dialog box when clicked. Push buttons are often used to open another dialog box. For other overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the push button, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the push button, in dialog units.
Width	Lets you specify the push button width, in dialog units.
Height	Lets you specify the push button height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.
Text	Lets you specify a string or a string variable that is used to set the initial text attribute for the push button's label.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`settext;gettext;enable;all_dyn_functions;;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`PUSHBUTTON;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

SPINCONTROL statement (for dynamic dialog boxes)

SPINCONTROL Left, Top, Width, Height, .Identifier

This statement adds a spin control to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the spin control, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the spin control, in dialog units.
Width	Lets you specify the spin control width, in dialog units.
Height	Lets you specify the spin control height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;setincrement;setprecision;setvalue;getprecision;getvalue;setdoublemode;setminrange;setmaxrange;getminrange;getmaxrange;enable;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`SPINCONTROL;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

STATUS statement (for dynamic dialog boxes)

STATUS .Identifier

This statement adds a status bar to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`settext;gettext;setstyle;getstyle;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`STATUS;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

TEXT statement (for dynamic dialog boxes)

TEXT Left, Top, Width, Height, .Identifier, Text

This statement adds a text label to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the text, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the text, in dialog units.
Width	Lets you specify the text label width, in dialog units.
Height	Lets you specify the text label height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.
Text	Lets you specify a string or a string variable that is used to set the text label's initial text attribute.

Note

- You can use the **SETSTYLE** function to change the properties of a text label. You can make the text label appear sunken, place a border around the text label, or align the text on the text label.

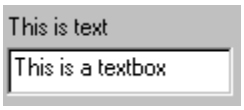
{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`all_dyn_functions;settext;gettext;enable;;;',0,"Defaultoverview",)} Dynamic dialog functions

{button ,AL(`TEXT;;;;;',1,"Defaultoverview",)} Syntax for static dialog boxes

TEXTBOX statement (for dynamic dialog boxes)

TEXTBOX Left, Top, Width, Height, .Identifier



This statement adds a text box to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the text box in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the text box in dialog units.
Width	Lets you specify the text label bounding box width in dialog units.
Height	Lets you specify the text label bounding box height in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

Note

- You can use the **SETSTYLE** function to specify how to display text in a filled single-line text box. Text can either scroll to the left or wrap to a new line in the text box.

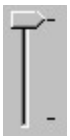
{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`all_dyn_functions;settext;gettext;enable;;;',0,"Defaultoverview",)} **Dynamic dialog functions**

{button ,AL(`TEXTBOX;;;;;',1,"Defaultoverview",)} **Syntax for static dialog boxes**

VSLIDER statement (for dynamic dialog boxes)

VSLIDER Left, Top, Width, Height, .Identifier



This statement adds a vertical slider control to a dynamic dialog box. For overview information about using this dialog control, see its static syntax topic.

Parameter	Description
Left	Lets you specify the distance from the inside of the dialog box's left border to the left side of the slider, in dialog units.
Top	Lets you specify the distance from the bottom of the dialog box's title bar to the top of the slider, in dialog units.
Width	Lets you specify the slider width, in dialog units.
Height	Lets you specify the slider height, in dialog units.
Identifier	Lets you specify a string variable that identifies the dynamic dialog control. The period (**.**) before the identifier is required.

{button ,AL(`corel_script_dialog_control_dynamic;;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`settick;gettick;setincrement;getincrement;setvalue;getvalue;setmaxrange;getmaxrange;setminrange;getminrange;enable;all_dyn_functions;',0,"Defaultoverview",)} **Dynamic dialog functions**

{button ,AL(`VSLIDER;;;;;',1,"Defaultoverview",)} **Syntax for static dialog boxes**

ADDITEM function

DialogID**.**Identifier**.ADDITEM Item**, Index

This function adds an item to a Corel SCRIPT list box dialog control. However, this function does not alter an array used to provide a list of items in a list box. Click the **Used with the following controls button** below for a list of Corel SCRIPT list box controls.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the list box is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the list box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Item	A string expression that specifies an item to add to a Corel SCRIPT list box.
Index	A numeric expression that specifies the position of the item in the list box. This value ranges from 1 to the current number of items in the list + 1. If not specified, the item is added as the last item in the list box list.

Note

- This function can only be used in a Dialog Event Handler subroutine.
- Use the **REMOVEITEM** function to remove a list box item.

{button ,AL(`lb_funs;setarray;',0,"Defaultoverview",)} Related Topics

{button ,AL(`listbox_dyn;ddlistbox_dyn;ddcombobox_dyn;combobox_dyn;imagelistbox_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

CLOSEDIALOG function

DialogID**.CLOSEDIALOG Value**

This function closes a dynamic dialog box.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Value	A numeric expression that specifies how to close the dialog box. Setting **Value** to 2 closes a dialog box and discards the settings (like pressing a Cancel button). Setting **Value** to anything but 2 closes the dialog box and assigns the settings within in it (like pressing an OK button). This value is also passed by **DIALOG** when it is used as a function to display a dialog box. For example: `ReturnValue% = DIALOG (Dialog1)` If you set **Value** to 3 and use the above example, **ReturnValue** equals 3 if the **CLOSEDIALOG** function closes the dialog box. See **Example 9** for more information.

Note

- This function, when called, does not immediately close a dialog box; that is, the dialog box is closed only after the Dialog Event Handler subroutine has terminated.

{button ,AL(`okbutton_dyn;cancelbutton_dyn;begin_end_dialog_dyn;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`begin_end_dialog_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

ENABLE function

DialogID**.**Identifier**.ENABLE Boolean**

This function enables and disables a specified dynamic dialog box control.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Boolean	Lets you specify whether to enable or disable the dynamic dialog box control. Set to TRUE (-1) to enable the control; otherwise, set to FALSE (0) to disable the control.

Note

- This function can only be used within the Dialog Event Handler subroutine. The dynamic dialog box that the control is used in must already be running to use this function.
- By default, all controls are enabled until you use the ENABLE function to disable them.

{button ,AL(`dialog_dyn;;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`BITMAPBUTTON_dyn;CANCELBUTTON_dyn;CHECKBOX_dyn;COMBOBOX_dyn;DDCOMBOBOX_dyn;DDLISTBOX_dyn;GROUPBOX_dyn;HELPBUTTON_dyn;HSLIDER_dyn;IMAGELISTBOX_dyn;LISTBOX_dyn;OKBUTTON_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;PUSHBUTTON_dyn;SPINCONTROL_dyn;TEXT_dyn;TEXTBOX_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

GETBITMAPHEIGHT function

ReturnValue = DialogID**.**Identifier**.GETBITMAPHEIGHT ( )**

Returns the height of a bitmap file used in an image box, in pixels.

Return Value

The numeric variable that is assigned the height of a bitmap in an image box, in pixels.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the image box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`GETBITMAPHEIGHT;GETBITMAPWIDTH;SETBITMAPOFFSET;',0,"Defaultoverview",)} Related Topics

{button ,AL(`IMAGE_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

GETBITMAPWIDTH function

ReturnValue = DialogID**.**Identifier**.GETBITMAPWIDTH ( )**

Returns the width of a bitmap file used in an image box, in pixels.

Return Value

The numeric variable that is assigned the width of a bitmap in an image box, in pixels.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the image box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`GETBITMAPHEIGHT;GETBITMAPWIDTH;SETBITMAPOFFSET;',0,"Defaultoverview",)} Related Topics

{button ,AL(`IMAGE_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

GETHEIGHT function

For dynamic dialog boxes:
ReturnValue = DialogID**.GETHEIGHT ( )**

For dynamic dialog box controls:
ReturnValue = DialogID**.**Identifier**.GETHEIGHT ( )**

Returns a specified dynamic dialog box or control's height, in dialog units.

Return Value

The numeric variable that is assigned the dynamic dialog box, or control's height attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`move;gettopposition;getleftposition;getheight;getwidth;',0,"Defaultoverview",)} Related Topics

{button ,AL(`begin_end_dialog_dyn;BITMAPBUTTON_dyn;CANCELBUTTON_dyn;CHECKBOX_dyn;COMBOBOX_dyn;DDCOMBOBOX_dyn;DDLISTBOX_dyn;GROUPBOX_dyn;HELPBUTTON_dyn;HSLIDER_dyn;IMAGE_dyn;IMAGELISTBOX_dyn;LISTBOX_dyn;OKBUTTON_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;PUSHBUTTON_dyn;SPINCONTROL_dyn;TEXT_dyn;TEXTBOX_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

GETHELPINDEX function

ReturnValue = DialogID**.**Identifier**.GETHELPINDEX ( )**

Returns a specified dynamic dialog help button's topic ID number attribute.

Return Value

The numeric variable that is assigned the help button's topic ID number attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the help button is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the help button. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`gethelppath;sethelppath;gethelpindex;sethelpindex;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`helpbutton_dyn;;;;;',0,"Defaultoverview",)} **Used with the following controls**

GETHELPPATH function

ReturnValue = DialogID**.**Identifier**.GETHELPPATH ( )**

Returns a specified dynamic dialog help button's corresponding help file and path.

Return Value

The string variable that is assigned the help button's help file and path attribute. The return value includes both the help file's filename and path.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the help button is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the help button. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`gethelppath;sethelppath;gethelpindex;sethelpindex;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`helpbutton_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

GETID function

ReturnValue = DialogID**.**Identifier**.GETID ( )**

Returns a specified dynamic dialog box control's position in the dialog box's definition statements. The first listed control is identified as 1, the second listed control is identified as 2, and so on. Although, OPTIONGROUP is not a control, it still uses a position value. This function is most useful in identifying the dialog control that generates a dialog event, especially in cases in which the order of the control's in a dialog box's definition under goes editing changes.

Return Value

The numeric variable that is assigned the dynamic dialog box control's position in the dialog box definition.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`Dialog Event Handler subroutine syntax;',0,"Defaultoverview",)} Related Topics

GETIMAGE function

ReturnValue = DialogID**.**Identifier**.GETIMAGE ( )**

Returns a specified dynamic dialog image control's image filename and path.

Return Value

The string variable that is assigned the return value corresponding to the dynamic dialog image control's image attribute. The return value includes the image's filename and path.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the image control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog image control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`setimage;getimage;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`image_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

GETINCREMENT function

ReturnValue = DialogID**.**Identifier**.GETINCREMENT ( )**

Returns a specified dynamic dialog increment value.

Return Value

The numeric variable that is assigned the dynamic dialog box control's increment attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`getincrement;setincrement;step;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`hslider_dyn;progress_dyn;spincontrol_dyn;vslider_dyn;;',0,"Defaultoverview",)} Used with the following controls

GETITEM function

ReturnString = DialogID**.**Identifier**.GETITEM (Index)**

Returns the item name of a specified dynamic dialog list box's item index number. Click the **Used with the following controls button** below for a list of Corel SCRIPT list box controls.

Return Value

The string variable that is assigned an item name corresponding to a specified dynamic dialog list box's item index number.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the list box is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the list box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Index	A numeric expression that specifies the position of the item in the list box. This value ranges from 1 to the current number of items in the list. The first item is 1, the second item is 2, and so on.

Note

- This function can only be used in a Dialog Event Handler subroutine.
- Use the **GETITEMCOUNT** function for the number of items in a list box.

{button ,AL(`lb_funs;setarray;',0,"Defaultoverview",)} Related Topics

{button ,AL(`listbox_dyn;ddlistbox_dyn;ddcombobox_dyn;combobox_dyn;imagelistbox_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

GETITEMCOUNT function

ReturnValue = DialogID**.**Identifier**.GETITEMCOUNT ( )**

Returns the number of items in a specified dynamic dialog dynamic dialog list box control. Click the **Used with the following controls button** below for a list of Corel SCRIPT list box controls.

Return Value

The numeric variable that is assigned the number of items in a specified dynamic dialog dynamic dialog list box control.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the list box is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the list box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

Note

- This function can only be used in a Dialog Event Handler subroutine.

{button ,AL(`lb_funs;setarray;',0,"Defaultoverview",)} Related Topics

{button ,AL(`listbox_dyn;ddlistbox_dyn;ddcombobox_dyn;combobox_dyn;imagelistbox_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

GETLEFTPOSITION function

For dynamic dialog boxes:
ReturnValue = DialogID**.GETLEFTPOSITION ( )**

For dynamic dialog box controls:
ReturnValue = DialogID**.**Identifier**.GETLEFTPOSITION ( )**

Returns a specified dynamic dialog box or control's horizontal distance, in dialog units.

Return Value

- For dynamic dialog boxes, the numeric variable that is assigned the distance in dialog units from the dialog box's left border to the left side of the monitor's display area.
- For dynamic dialog box controls, the numeric variable that is assigned the distance in dialog units from the inside of the dialog box's left border to the left side of the control.

Note

- This function returns -1 when used with a centered dialog box. See **BEGIN DIALOG** for more information about centering dialog boxes.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`move;gettopposition;getleftposition;getheight;getwidth;',0,"Defaultoverview",)} Related Topics

{button ,AL(`begin_end_dialog_dyn;BITMAPBUTTON_dyn;CANCELBUTTON_dyn;CHECKBOX_dyn;COMBOBOX_dyn;DDCOMBOBOX_dyn;DDLISTBOX_dyn;GROUPBOX_dyn;HELPBUTTON_dyn;HSLIDER_dyn;IMAGE_dyn;IMAGELISTBOX_dyn;LISTBOX_dyn;OKBUTTON_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;PUSHBUTTON_dyn;SPINCONTROL_dyn;TEXT_dyn;TEXTBOX_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

GETMAXRANGE function

ReturnValue = DialogID**.**Identifier**.GETMAXRANGE ( )**

Returns a specified dynamic dialog box control's maximum value.

Return Value

The numeric variable that is assigned the dynamic dialog box control's maximum value attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`getmaxrange;getminrange;setmaxrange;setminrange;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`hslider_dyn;progress_dyn;spincontrol_dyn;vslider_dyn;;',0,"Defaultoverview",)} **Used with the following controls**

GETMINRANGE function

ReturnValue = DialogID**.**Identifier**.GETMINRANGE ( )**

Returns a specified dynamic dialog box control's minimum value.

Return Value

The numeric variable that is assigned the dynamic dialog box control's minimum value attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`getmaxrange;getminrange;setmaxrange;setminrange;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`hslider_dyn;progress_dyn;spincontrol_dyn;vslider_dyn;;',0,"Defaultoverview",)} **Used with the following controls**

GETPRECISION function

ReturnValue = DialogID**.**Identifier**.GETPRECISION ( )**

Returns the number of decimals used with a dynamic dialog spin control's value attribute.

Return Value

The numeric variable that is assigned the number of decimals used to set a dynamic dialog spin control's value. Valid values range from 0 to 5, inclusive.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the spin control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog spin control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`setprecision;setprecision;setdoublemode;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`spincontrol_dyn;;;;;',0,"Defaultoverview",)} **Used with the following controls**

GETSELECT function

ReturnValue = DialogID**.**Identifier**.GETSELECT ( )**

Returns a specified dynamic dialog box control's selection attribute. This function returns the selection in list boxes and combo list boxes.

Return Value

- For list boxes, drop-down list boxes and image list boxes, a numeric variable that is assigned the array element selected.
- For combo boxes and drop-down combo boxes, a string variable that is assigned the text attribute in the text box portion of the control.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`settext;setselect;gettext;getselect;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`listbox_dyn;combobox_dyn;ddlistbox_dyn;ddcombobox_dyn;imagelistbox_dyn;;;',0,"Defaultoverview",)} Used with the following controls

GETSTYLE function

For dynamic dialog boxes:
ReturnValue = DialogID**.GETSTYLE ( )**

For dynamic dialog box controls:
ReturnValue = DialogID**.**Identifier**.GETSTYLE ( )**

Returns a specified dynamic dialog box or control's style attributes.

Return Value

The numeric variable that is assigned the dynamic dialog box or control's style attributes. This function returns the following values:

- Applies to dialog boxes and all dialog controls
  - 1 Visible(default)
  - 2 Invisible
- Applies to slider, option button, check box, text, and image controls only
  - 4 No border (default)
  - 8 Border (click for an example)
- Applies to list, drop-down list, combo, and drop-down combo boxes controls only
  - 4 No sorting (default)
  - 8 Sorts items in list boxes
- Applies to text and text box controls only
  - 16 Left Aligned (default)
  - 32 Right Aligned
  - 64 Centered

  Justification in a text box can only be enabled if the text wrapping style is enabled (1024).
- Applies to dialog boxes only
  - 16 No title bar options (default)
  - 32 Title bar options which include an icon, and minimize and maximize button
  - 64 Same options as 32 as well as allowing user to resize dialog box. Click for an example.
- Applies to bitmap button and image controls only
  - 16 Resize bitmap to fit control (default)
  - 32 Center bitmap in control
  - 64 Position bitmap in upper-left corner of control
- Applies to horizontal slider and vertical slider controls only
  - 16 For horizontal sliders, tick marks are placed below the slider and the slider indicator points down. For vertical sliders, tick marks are placed to the right of the slider and the slider indicator points right. This is the default setting
  - 32 For horizontal sliders, tick marks are placed above the slider and the slider indicator points up. For vertical sliders, tick marks are placed to the left of the slider and the slider indicator points left.
  - 64 Tick marks are placed on both sides of the slider and the slider indicator is a rectangle.
- Applies to text and image controls only
  - 128 Not sunken (default)
  - 256 Sunken (click for an example)
- Applies to text box controls only
  - 512 Text scrolls to the left when text box is filled (default)
  - 1024 Text wraps when text box is filled
- Applies to text box controls only
  - 2048 Text is entered as typed (default)
  - 4096 Text is entered in uppercase characters
  - 8192 Text is entered in lowercase characters

- Applies to text box controls only

  16384 Text is entered as typed (default)

  32768 Text is entered password mode (all characters are entered as asterisks).

  Enabling password mode in a text box disables the text wrapping, justification, and text case styles.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

Note

- By default, all dynamic dialog box control's style is set to Visible (1) except Text controls which are set to Visible (1) and Left Aligned (16).
- In cases where multiple attributes are returned, you can use the **AND** (bitwise) operator to determine specific attributes. The following example shows how to test if a return value variable includes the Sunken attribute:
  - `IF 128 AND ReturnValue THEN sunkenTest$ = "Yes" ELSE sunkenTest$ = "No"`
  - 128 is the Sunken attribute. The variable **sunkenTest** is assigned a string based on bitwise comparison. In this example, **sunkenTest** is assigned "Yes" if **ReturnValue** has the Sunken attribute.

{button ,AL(`SETVISIBLE;setstyle;getstyle;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`begin_end_dialog_dyn;BITMAPBUTTON_dyn;CANCELBUTTON_dyn;CHECKBOX_dyn;COMBOBOX_dyn;DDCOMBOBOX_dyn;DDLISTBOX_dyn;GROUPBOX_dyn;HELPBUTTON_dyn;HSLIDER_dyn;IMAGE_dyn;IMAGELISTBOX_dyn;LISTBOX_dyn;OKBUTTON_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;PUSHBUTTON_dyn;SPINCONTROL_dyn;TEXT_dyn;TEXTBOX_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

GETTEXT function

For dynamic dialog boxes:
ReturnValue = DialogID**.GETTEXT ( )**

For dynamic dialog box controls:
ReturnValue = DialogID**.**Identifier**.GETTEXT ( )**

Returns a specified dynamic dialog box or control's text or label attribute.

Return Value

The string variable that is assigned the dynamic dialog box or control's text or label attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`settext;setselect;gettext;getselect;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`BEGIN_END_DIALOG_dyn;CHECKBOX_dyn;GROUPBOX_dyn;OPTIONBUTTON_dyn;PUSHBUTTON_dyn;TEXT_dyn;TEXTBOX_dyn;STATUS_dyn;;;;;;',0,"Defaultoverview",)} **Used with the following controls**

GETTICK function

ReturnValue = DialogID**.**Identifier**.GETTICK ( )**

Returns the interval of tick marks on a horizontal or vertical slider control.

Return Value

A numeric variable that is assigned the interval of tick marks on a horizontal or vertical slider control. Assigning 0 as a return value indicates tick marks are not used except at the ends of the slider.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies a horizontal or vertical slider control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`settick;gettick;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`hslider;vslider;',0,"Defaultoverview",)} Used with the following controls

GETTIMER function

ReturnValue = DialogID**.GETTIMER ( )**

Returns a specified dynamic dialog box's timer value in milliseconds. A dialog box's timer begins to count down once the dialog box is initialized and can be used by the Dialog Event Handler subroutine to trigger a dialog event.

Return Value

A numeric variable that is assigned a dialog box's timer setting in milliseconds. One second is equal to 1000 milliseconds, 10 seconds is equal to 10,000 milliseconds, and one minute is equal to 60,000 milliseconds.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.

{button ,AL(`settimer;gettimer;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`BEGIN_END_DIALOG_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

GETTOPPOSITION function

For dynamic dialog boxes:
ReturnValue = DialogID**.GETTOPPOSITION ( )**

For dynamic dialog box controls:
ReturnValue = DialogID**.**Identifier**.GETTOPPOSITION ( )**

Returns a specified dynamic dialog box or control's vertical distance, in dialog units.

Return Value

- For dynamic dialog boxes, the numeric variable that is assigned the distance in dialog units from the dialog box's top border to the top side of the monitor's display area.
- For dynamic dialog box controls, the numeric variable that is assigned the distance from the bottom of the dialog box's title bar to the top of the control.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`move;gettopposition;getleftposition;getheight;getwidth;',0,"Defaultoverview",)} Related Topics

{button ,AL(`begin_end_dialog_dyn;BITMAPBUTTON_dyn;CANCELBUTTON_dyn;CHECKBOX_dyn;COMBOBOX_dyn;DDCOMBOBOX_dyn;DDLISTBOX_dyn;GROUPBOX_dyn;HELPBUTTON_dyn;HSLIDER_dyn;IMAGE_dyn;IMAGELISTBOX_dyn;LISTBOX_dyn;OKBUTTON_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;PUSHBUTTON_dyn;SPINCONTROL_dyn;TEXT_dyn;TEXTBOX_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

GETVALUE function

ReturnValue = DialogID**.**Identifier**.GETVALUE ( )**

Returns a specified dynamic dialog box control's value.

Return Value

The numeric variable that is assigned the dynamic dialog box control's selection or value attribute.

Dialog controls	Return values
CHECKBOX	Check boxes return the following values: 0 Cleared 1 Displays a check mark 2 Grayed checkbox. Filling a checkbox with gray indicates that a multiple selection contains a mix of property values. For example, selecting text that uses different fonts returns a mixed value. This value is only available when the specified checkbox is in **Three State mode**.
HSLIDER	A horizontal slider's indicator returns a whole number value (Long) between **SETMINRANGE** and **SETMAXRANGE**, inclusive. If the **SETMINRANGE** function is not used, the minimum value that can be returned is 0. If the **SETMAXRANGE** function is not used, the maximum value that can be returned is 100.
OPTIONBUTTON	Option buttons return the following values: 0 Disabled 1 Enabled
OPTIONGROUP	The OPTIONGROUP construct returns a whole number that corresponds to the option button selected by the user. The first option button in an option group is identified as 0, the second option button is identified as 1, and so on. The order of the option buttons is determined, not by their placement within the dialog box, but by the order in which they are listed in the script. You cannot use the **GETVALUE** function with an OPTIONGROUP once the dialog box is running.
SPINCONTROL	A spin control's value returns a number between **SETMINRANGE** and **SETMAXRANGE**, inclusive. If the **SETMINRANGE** function is not used, the minimum value that can be returned is -32,768. If the **SETMAXRANGE** function is not used, the maximum value that can be returned is 32,767. A spin control can return a fractional number if the **SETDOUBLEMODE** function is enabled. If **SETDOUBLEMODE** is enabled, a spin control's value range can take on the range of a Double data type. If **SETDOUBLEMODE** is enabled, the number of decimal places that are used depends on the **SETPRECISION** function.
VSLIDER	A vertical slider's indicator returns a whole number value (Long) between **SETMINRANGE** and **SETMAXRANGE**, inclusive. If the **SETMINRANGE** function is not used, the minimum value that can be returned is 0. If the **SETMAXRANGE** function is not used, the maximum value that can be returned is 100.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

{button ,AL(`setvalue;getvalue;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`CHECKBOX_dyn;HSLIDER_dyn;optiongroup_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;SPINCONTROL_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

GETWIDTH function

For dynamic dialog boxes:
ReturnValue = DialogID**.GETWIDTH ( )**

For dynamic dialog box controls:
ReturnValue = DialogID**.**Identifier**.GETWIDTH ( )**

Returns a specified dynamic dialog box or control's width, in dialog units.

Return Value

The numeric variable that is assigned the dynamic dialog box or control's width attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

Note

{button ,AL(`move;gettopposition;getleftposition;getheight;getwidth;',0,"Defaultoverview",)} Related Topics

{button ,AL(`begin_end_dialog_dyn;BITMAPBUTTON_dyn;CANCELBUTTON_dyn;CHECKBOX_dyn;COMBOBOX_dyn;DDCOMBOBOX_dyn;DDLISTBOX_dyn;GROUPBOX_dyn;HELPBUTTON_dyn;HSLIDER_dyn;IMAGE_dyn;IMAGELISTBOX_dyn;LISTBOX_dyn;OKBUTTON_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;PUSHBUTTON_dyn;SPINCONTROL_dyn;TEXT_dyn;TEXTBOX_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

MOVE function

For dynamic dialog boxes:
DialogID**.MOVE Left, Top**, Width, Height

For dynamic dialog box controls:
DialogID**.**Identifier**.MOVE Left, Top**, Width, Height

This function moves and resizes a dynamic dialog box, or the controls within a dynamic dialog box.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Left	For dialog boxes, specifies the distance in dialog units from the dialog box's left border to the left side of the monitor's display area. For controls, specifies the distance in dialog units from the inside of the dialog box's left border to the left side of the control.
Top	For dialog boxes, specifies the distance in dialog units from the dialog box's top border to the top side of the monitor's display area. For controls, specifies the distance from the bottom of the dialog box's title bar to the top of the check box, in dialog units.
Width	Lets you specify the dialog box or control width in dialog units. This is optional parameter, and if omitted, the **Height** parameter must also be omitted. Omitting this parameter leaves the dialog box or control size unchanged.
Height	Lets you specify the dialog box or control height in dialog units. This is optional parameter, and if omitted, the **Width** parameter must also be omitted. Omitting this parameter leaves the dialog box or control size unchanged.

{button ,AL(`move;gettopposition;getleftposition;getheight;getwidth;',0,"Defaultoverview",)} Related Topics

{button ,AL(`begin_end_dialog_dyn;BITMAPBUTTON_dyn;CANCELBUTTON_dyn;CHECKBOX_dyn;COMBOBOX_dyn;DDCOMBOBOX_dyn;DDLISTBOX_dyn;GROUPBOX_dyn;HELPBUTTON_dyn;HSLIDER_dyn;IMAGE_dyn;IMAGELISTBOX_dyn;LISTBOX_dyn;OKBUTTON_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;PUSHBUTTON_dyn;SPINCONTROL_dyn;TEXT_dyn;TEXTBOX_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

REMOVEITEM function

DialogID**.**Identifier**.REMOVEITEM Index**

This function removes an item from a Corel SCRIPT list box dialog control. However, this function does not alter an array used to provide a list of items in a list box. Click the **Used with the following controls button** below for a list of Corel SCRIPT list box controls.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the list box is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the list box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Index	A numeric expression that specifies the position of the item in the list box to remove. This value ranges from 1 to the current number of items in the list. The first item is 1, the second item is 2, and so on.

Note

- This function can only be used in a Dialog Event Handler subroutine.
- Use the **RESET** function to remove all the items from a list box.
- Use the **ADDITEM** function to add a list box item.

{button ,AL(`lb_funs;setarray;',0,"Defaultoverview",)} Related Topics

{button ,AL(`listbox_dyn;ddlistbox_dyn;ddcombobox_dyn;combobox_dyn;imagelistbox_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

RESET function

DialogID**.**Identifier**.RESET**

This function removes all the items from a Corel SCRIPT list box dialog control. However, this function does not alter an array used to provide a list of items in a list box. Click the **Used with the following controls button** below for a list of Corel SCRIPT list box controls.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the list box is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the list box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

Note

- This function can only be used in a Dialog Event Handler subroutine.
- Use the **REMOVEITEM** function to only remove one item from a list box.
- Use the **ADDITEM** function to add a list box item.

{button ,AL(`lb_funs;setarray;',0,"Defaultoverview",)} Related Topics

{button ,AL(`listbox_dyn;ddlistbox_dyn;ddcombobox_dyn;combobox_dyn;imagelistbox_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

SETARRAY function

DialogID**.**Identifier**.SETARRAY Array**

Sets a specified dynamic dialog box control's array attribute. Arrays are used to create lists to display in list boxes and combo list boxes, or set a bitmap button's image properties.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Array	Lets you specify a one-dimension array containing a string list of items that appears in the control. Arrays used with the **IMAGELISTBOX** or **BITMAPBUTTON** control must contain a string list of Windows' bitmaps (.BMP and .RLE files) and their paths. When using this function with a bitmap button control, only the first four array elements are used. Each array element corresponds to the bitmap button's state. The button's state determines which bitmap in the array to reference and display: Array element 1 — Normal state Array element 2 — Depressed state Array element 3 — Focus state Array element 4 — Disable state If an array element entry is not available (for example, bad file name), the button is filled with black for the state corresponding to the unavailable element entry. If the array does not have an element for index numbers 1, 2, 3, or 4, the button is filled with black for the state corresponding to the missing element index number, along with buttons that follow it in the array. For example, if array element 2 is missing, the bitmap button is filled with black for the Depressed, Focus, and Disable states.

Note

- You can also use the **ADDITEM** and **REMOVEITEM** functions to add and remove items from a list box control without altering the array used to provide the list of items in a list box.
- When using this function with a bitmap button control and an error occurs, the bitmap button control is filled with black.

{button ,AL(`setselect;getselect;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`listbox_dyn;combobox_dyn;ddlistbox_dyn;ddcombobox_dyn;bitmapbutton_dyn;imagelistbox_dyn;;;',0,"Defaultoverview",)} **Used with the following controls**

SETBITMAPOFFSET function

DialogID**.**Identifier**.SETBITMAPOFFSET Left, Top**, Width, Height

Sets the portion of a bitmap to display in an image box. Using this function allows you to use one bitmap to display more than one item. See the example for more information.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the image box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Left	Lets you specify the horizontal position of the upper-left corner of the portion of the bitmap to display, in pixels.
Top	Lets you specify the vertical position of the upper-left corner of the portion of the bitmap to display, in pixels.
Width	Lets you specify the width of the portion of the bitmap to display, in pixels.
Height	Lets you specify the height of the portion of the bitmap to display, in pixels.

Note

- By omitting the last two parameters and setting **Left** and **Top** to 0, the **SETBITMAPOFFSET** function resets the image box to display the entire bitmap.

{button ,AL(`GETBITMAPHEIGHT;GETBITMAPWIDTH;SETBITMAPOFFSET;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`IMAGE_dyn;;;;;',0,"Defaultoverview",)} **Used with the following controls**

SETDOUBLEMODE function

DialogID**.**Identifier**.SETDOUBLEMODE Boolean**

Sets the type of numeric data a dynamic dialog spin control can hold.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the spin control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog spin control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Boolean	Lets you specify whether a spin control can hold Long type data or Double type data. Set to TRUE (-1) to enable a specified spin control to hold Double type data. Set to FALSE (0) to enable a specified spin control to hold Long type data. If this function is not used, by default, a spin control in a dynamic dialog box can only hold a Long value.

Note

- When a spin control switches from Double type data to Long type data, current spin control decimal values are truncated.
- If **SETDOUBLEMODE** is enabled, the number of decimal places a spin control can use depends on the **SETPRECISION** function.

{button ,AL(`setprecision;getprecision;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`spincontrol_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

SETHELPINDEX function

DialogID**.**Identifier**.SETHELPINDEX Value**

Within a dynamic dialog box, this function sets the topic ID number associated with a specified help button.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the help button is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the help button. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Value	A numeric expression that specifies a valid topic ID number corresponding to the topic WinHelp displays whenever the help button is clicked.

Note

- If a topic ID number is invalid or not specified, and the help button is clicked, WinHelp opens an information dialog box notifying the user the topic is not available.

{button ,AL(`gethelppath;sethelppath;gethelpindex;sethelpindex;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`helpbutton_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

SETHELPPATH function

DialogID**.**Identifier**.SETHELPPATH Text**

Within a dynamic dialog box, this function sets the Windows help file associated with a specified help button.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the help button is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the help button. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Text	A string expression that specifies a valid Windows help file. The string expression must include the help file's filename and path.

Note

- If the **SETHELPPATH** function is not used with a help button in a script, and the help button is clicked, Corel SCRIPT opens an information dialog box notifying the user that a help file has not been specified.

{button ,AL(`gethelppath;sethelppath;gethelpindex;sethelpindex;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`helpbutton_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

SETIMAGE function

DialogID**.**Identifier**.SETIMAGE Text**

Sets a specified dynamic dialog image control's image attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the image control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog image control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Text	A string expression that specifies a dynamic dialog image control's image attribute. The string expression must include the image's filename and path. Windows bitmaps (.BMP and .RLE files) are valid image files.

Note

- If the image can't be found on the user's system, an X fills the image control.

{button ,AL(`setimage;getimage;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`image_dyn;;;;;',0,"Defaultoverview",)} **Used with the following controls**

SETINCREMENT function

DialogID**.**Identifier**.SETINCREMENT Value**

Sets a specified dynamic dialog increment value.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Value	A numeric expression that specifies a dynamic dialog box control's increment value.

Dialog controls	Value settings
HSLIDER	A horizontal slider's increment value is set to a whole number. The default increment value is 10. A horizontal slider's increment value determines the amount the slider indicator moves when the user clicks along the slider bar, or presses the arrow keys on the keyboard when the slider control has focus. **Note:** Clicking along the slider moves the indicator twice the increment value.
PROGRESS	A progress indicator's increment value is set to a whole number and determines the amount the completion gauge changes for each progress step. The default increment value is 10. For more information, see the **STEP** function.
SPINCONTROL	A spin control's increment value determines the amount the control value changes when the spin control arrows are clicked. The default increment value is 1. You can set a spin control's increment value to a fractional number if the **SETDOUBLEMODE** function is enabled. If **SETDOUBLEMODE** is enabled, the number of decimal places a spin control can use depends on the **SETPRECISION** function.
VSLIDER	A vertical slider's increment value is set to a whole number. The default increment value is 10. A vertical slider's increment value determines the amount the slider indicator moves when the user clicks along the slider bar, or presses the arrow keys on the keyboard when the slider control has focus. **Note:** Clicking along the slider moves the indicator twice the increment value.

{button ,AL(`getincrement;setincrement;step;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`hslider_dyn;progress_dyn;spincontrol_dyn;vslider_dyn;;',0,"Defaultoverview",)} Used with the following controls

SETMAXRANGE function

DialogID**.**Identifier**.SETMAXRANGE Long**

Sets a specified dynamic dialog box control's maximum value.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Long	A numeric expression that specifies a dynamic dialog box control's maximum value. This value must be greater than the control's minimum value. See **SETMINRANGE** for more information.

Note

- By default, controls that use the **SETMAXRANGE** function have a default maximum value of 100, except the spin control, which is set to 32,767.
- You can set a spin control's range value to a fractional number if the **SETDOUBLEMODE** function is enabled. If **SETDOUBLEMODE** is enabled, a spin control's value range can take on the range of a Double data type. If **SETDOUBLEMODE** is enabled, the number of decimal places a spin control can use depends on the **SETPRECISION** function.

{button ,AL(`getmaxrange;getminrange;setmaxrange;setminrange;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`hslider_dyn;progress_dyn;spincontrol_dyn;vslider_dyn;;',0,"Defaultoverview",)} Used with the following controls

SETMINRANGE function

DialogID**.**Identifier**.SETMINRANGE Long**

Sets a specified dynamic dialog box control's minimum value.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Long	A numeric expression that specifies a dynamic dialog box control's minimum value. This value must be less than the control's maximum value. See **SETMAXRANGE** for more information.

Note

- By default, controls that use the **SETMINRANGE** function have a default minimum value of 0, except the spin control, which is set to -32,768.
- You can set a spin control's range value to a fractional number if the **SETDOUBLEMODE** function is enabled. If **SETDOUBLEMODE** is enabled, a spin control's value range can take on the range of a Double data type. If **SETDOUBLEMODE** is enabled, the number of decimal places a spin control can use depends on the **SETPRECISION** function.
- The minimum value for the progress bar must be greater than or equal to 0.

{button ,AL(`getmaxrange;getminrange;setmaxrange;setminrange;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`hslider_dyn;progress_dyn;spincontrol_dyn;vslider_dyn;;',0,"Defaultoverview",)} Used with the following controls

SETPRECISION function

DialogID**.**Identifier**.SETPRECISION Long**

Set the number of decimals used with a dynamic dialog spin control's value attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the spin control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog spin control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Long	A numeric expression that specifies the number of decimals to use to set a dynamic dialog spin control's value. Valid values range from 0 to 5, inclusive. The default value is 2.

Note

- If the specified spin control is not in double data type mode (**SETDOUBLEMODE**), the **SETPRECISION** function is ignored.
- Reducing the number of decimals used in a spin control rounds the spin control's active value.

{button ,AL(`setprecision;setprecision;setdoublemode;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`spincontrol_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

SETSELECT function

DialogID**.**Identifier**.SETSELECT Value**

Sets a specified dynamic dialog box control's selection attribute. This function is often used to create a default selection in list boxes and combo list boxes.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Value	For list boxes, drop-down list boxes, and image list boxes, a numeric expression that specifies an array element that corresponds to the list selection to select. For combo boxes and drop-down combo boxes, a string expression that specifies the text attribute in the text box portion of the control. You can select an array element by using the following syntax: `Identifier.SETSELECT(array(x))` where **x** is an array element index number. If you set the text attribute in a combo box's text box portion equal to a value in the list box portion, the value in the list box portion is also selected.

{button ,AL(`settext;setselect;gettext;getselect;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`listbox_dyn;combobox_dyn;ddlistbox_dyn;ddcombobox_dyn;imagelistbox_dyn;;;',0,"Defaultoverview",)} Used with the following controls

SETSTYLE function

For dynamic dialog boxes:
DialogID**.SETSTYLE Value**

For dynamic dialog box controls:
DialogID**.**Identifier**.SETSTYLE Value**

Sets a specified dynamic dialog box or a control's attributes.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Value	A numeric expression that specifies a dynamic dialog box or a control's style attributes. **Applies to dialog boxes and all dialog controls** 1 Visible(default) 2 Invisible **Applies to slider, option button, check box, text, and image controls only** 4 No border (default) 8 Border (click for an example) **Applies to list, drop-down list, combo, and drop-down combo boxes controls only** 4 No sorting (default) 8 Sorts items in list boxes **Applies to text and text box controls only** 16 Left Aligned (default) 32 Right Aligned 64 Centered Justification in a text box can only be enabled if the text wrapping style is enabled (1024). **Applies to dialog boxes only** 16 No title bar options (default) 32 Title bar options which include an icon, and minimize and maximize button 64 Same options as 32 as well as allowing user to resize dialog box. Click for an example. **Applies to bitmap button and image controls only** 16 Resize bitmap to fit control (default) 32 Center bitmap in control 64 Position bitmap in upper-left corner of control **Applies to horizontal slider and vertical slider controls only** 16 For horizontal sliders, tick marks are placed below the slider and the slider indicator points down. For vertical sliders, tick marks are placed to the right of the slider and the slider indicator points right. This is the default setting 32 For horizontal sliders, tick marks are placed above the slider and the slider indicator points up. For vertical sliders, tick marks are placed to the left of the slider and the slider indicator points left. 64 Tick marks are placed on both sides of the slider and the slider indicator is a rectangle. **Applies to text and image controls only** 128 Not sunken (default) 256 Sunken (click for an example) **Applies to text box controls only** 512 Text scrolls to the left when text box is filled (default) 1024 Text wraps when text box is filled **Applies to text box controls only** 2048 Text is entered as typed (default) 4096 Text is entered in uppercase characters 8192 Text is entered in lowercase characters **Applies to text box controls only** 16384 Text is entered as typed (default) 32768 Text is entered password mode (all characters are entered as asterisks).

Enabling password mode in a text box disables the text wrapping, justification, and text case styles.

Note

- The Border, Sunken, and Title bar styles can only be applied outside the Dialog Event Handler subroutine; that is, before the **Dialog** statement which is used to display the dialog box. See the example for more information. You cannot use these styles once a dialog box is displayed.
- You should also set the dialog box's title bar and resizing option outside of the Dialog Event Handler subroutine.
- You can also use the **SETVISIBLE** function to hide and display dialog boxes and controls.
- To apply more than one style to a control use the **OR** operator. For example, to right align and place a border around a text control, use the **SETSTYLE** function in the following manner:

```
Text1.SETSTYLE 32 OR 8
```

{button ,AL(`SETVISIBLE;setstyle;getstyle;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`BITMAPBUTTON_dyn;CANCELBUTTON_dyn;CHECKBOX_dyn;COMBOBOX_dyn;DDCOMBOBOX_dyn;DDLISTBOX_dyn;GROUPBOX_dyn;HELPBUTTON_dyn;HSLIDER_dyn;IMAGE_dyn;IMAGELISTBOX_dyn;LISTBOX_dyn;OKBUTTON_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;PUSHBUTTON_dyn;SPINCONTROL_dyn;STATUS_dyn;TEXT_dyn;TEXTBOX_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

SETTEXT function

For dynamic dialog boxes:
DialogID**.SETTEXT(Text)**

For dynamic dialog box controls:
DialogID**.**Identifier**.SETTEXT Text**

Sets a specified dynamic dialog box or a control's text or label attribute.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Text	A string expression that specifies a dynamic dialog box or control's text or label attribute.

{button ,AL(`settext;setselect;gettext;getselect;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`BEGIN_END_DIALOG_dyn;CHECKBOX_dyn;GROUPBOX_dyn;OPTIONBUTTON_dyn;PUSHBUTTON_dyn;TEXT_dyn;TEXTBOX_dyn;STATUS_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

SETTHREESTATE function

DialogID**.**Identifier**.SETTHREESTATE Boolean**

Sets a checkbox in a dynamic dialog check box to two or three states.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Boolean	Lets you specify whether a checkbox control can be set to two (Enabled and Disabled) or three states (Enabled, Disabled, and Mixed). Set to TRUE (-1) to enable three states. Set to FALSE (0) to enable two states.

Note

- By default a checkbox in a dynamic dialog box is set to three states.

{button ,AL(`getvalue;setvalue;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`checkbox_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

SETTICK function

DialogID**.**Identifier**.SETTICK Value**

Sets the interval of tick marks on a horizontal or vertical slider control.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Value	A numeric expression that specifies the interval of tick marks on a slider control. For example, setting this parameter to 5, sets a tick mark every 5 units in the slider's range.

Note

- If a slider control does not use the **SETTICK** function, tick marks are only placed at the ends of the slider.
- Setting **SETTICK** to 0 places tick marks at the ends of the slider only.

{button ,AL(`settick;gettick;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`hslider;vslider;',0,"Defaultoverview",)} **Used with the following controls**

SETTIMER function

DialogID**.SETTIMER Value**

Sets a specified dynamic dialog box's timer value in milliseconds. A dialog box's timer begins to count down once the specified dialog box is initialized and can be used by the Dialog Event Handler subroutine to trigger a dialog event.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Value	A numeric expression that specifies a dialog box's timer setting in milliseconds. One second is equal to 1000 milliseconds, 10 seconds is equal to 10,000 milliseconds, and one minute is equal to 60,000 milliseconds. This numeric value must be equal to or greater than 0. Setting this parameter to 0 disables the Time out event in the Dialog Event Handler subroutine. By default, the Time out event is disabled in dynamic dialog boxes.

Note

- Click for more information about using the timer.

{button ,AL(`settimer;gettimer;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`BEGIN_END_DIALOG_dyn;;;;;',0,"Defaultoverview",)} Used with the following controls

SETVALUE function

DialogID**.**Identifier**.SETVALUE Value**

Sets a specified dynamic dialog box control value.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the control is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Value	A numeric expression that specifies a dynamic dialog box control's value.

Dialog controls	Value settings
CHECKBOX	Check boxes can be set to the following values: 0 Cleared 1 Displays a check mark 2 Grayed checkbox. Filling a checkbox with gray indicates that a multiple selection contains a mix of property values. For example, selecting text that uses different fonts returns a mixed value. This value is only available when the specified checkbox is in Three State mode.
HSLIDER	A horizontal slider's indicator can be set to a whole number value (Long) between **SETMINRANGE** and **SETMAXRANGE**, inclusive. If the **SETMINRANGE** function is not used, then the default minimum value is set to 0. If the **SETMAXRANGE** function is not used, then the default maximum value is set to 100. The default value for a horizontal slider is 0 if the **SETVALUE** function is not used.
OPTIONBUTTON	Option buttons can be set to the following values: 0 Disabled (default) 1 Enabled (only one option button in an option group can be enabled at once) This value can be overridden by using the **SETVALUE** function with the option button's option group.
OPTIONGROUP	Sets the option button to enable. The first option button in a group corresponds to 0, the second button corresponds to 1, and so on. You cannot use the **SETVALUE** function with an OPTIONGROUP once the dialog box is running.
PROGRESS	A progress indicator's completion gauge can be set to a whole number value (Long) between **SETMINRANGE** and **SETMAXRANGE**, inclusive. If the **SETMINRANGE** function is not used, the default minimum value is set to 0. If the **SETMAXRANGE** function is not used, the default maximum value is set to 100. The default value for a progress indicator is 0 if the **SETVALUE** function is not used.
SPINCONTROL	A spin control's value can be set to a whole number value (Long) between **SETMINRANGE** and **SETMAXRANGE**, inclusive. If the **SETMINRANGE** function is not used, the default minimum value is set to -32,768. If the **SETMAXRANGE** function is not used, the default maximum value is set to 32,767. You can set a spin control's value to a fractional number if the **SETDOUBLEMODE** function is enabled. If **SETDOUBLEMODE** is enabled, a spin control's value range can take on the range of a Double data type. If **SETDOUBLEMODE** is enabled, the number of decimal places a spin control can use depends on the **SETPRECISION** function. The default value for a spin control is 1 if the **SETVALUE** function is not used.
VSLIDER	A vertical slider's indicator can be set to a whole number value (Long) between **SETMINRANGE** and **SETMAXRANGE**, inclusive. If the **SETMINRANGE** function is not used, then the default minimum value is set to 0. If the **SETMAXRANGE** function is not used, then the default maximum value is set to 100. The default value for a vertical slider is 0 if the **SETVALUE** function is not used.

{button ,AL(`setvalue;getvalue;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`CHECKBOX_dyn;HSLIDER_dyn;optiongroup_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;SPINCONTROL_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

SETVISIBLE function

For dynamic dialog boxes:
DialogID**.SETVISIBLE Boolean**

For dynamic dialog box controls:
DialogID**.**Identifier**.SETVISIBLE Boolean**

Displays or hides a specified dynamic dialog box or dynamic dialog box control.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.
Boolean	Lets you specify whether to hide or display a dynamic dialog box or dynamic dialog box control. Set to TRUE (-1) to display; set to FALSE (0) to hide.

Note

- You should consider using this function to hide a dialog box which opens a secondary dialog box. Once the secondary dialog box closes you can then display the first dialog box again. Click the example button for more information.
- If a script hides a dialog box and has no way of redisplaying it, script execution is frozen. The only way to exit frozen script execution is to close the Corel SCRIPT session. To close the Corel SCRIPT session in Windows 95, press CTRL+ALT+DEL, select the Corel SCRIPT session, and click End Task. In Windows NT, press CTRL+ALT+DEL, click Task Manager, select the Corel SCRIPT session, and click End Task.
- You can use the **GETSTYLE** function to return a dynamic dialog box's or control's visible setting.

{button ,AL(`setSTYLE;getSTYLE;;;;',0,"Defaultoverview",)} Related Topics

{button ,AL(`BITMAPBUTTON_dyn;CANCELBUTTON_dyn;CHECKBOX_dyn;COMBOBOX_dyn;DDCOMBOBOX_dyn;DDLISTBOX_dyn;GROUPBOX_dyn;HELPBUTTON_dyn;HSLIDER_dyn;IMAGE_dyn;IMAGELISTBOX_dyn;LISTBOX_dyn;OKBUTTON_dyn;OPTIONBUTTON_dyn;PROGRESS_dyn;PUSHBUTTON_dyn;SPINCONTROL_dyn;STATUS_dyn;TEXT_dyn;TEXTBOX_dyn;VSLIDER_dyn;;;;;;',0,"Defaultoverview",)} Used with the following controls

STEP function

DialogID**.**Identifier**.STEP**

This function adds to a specified progress indicator's value, the value of it's increment setting (**SETINCREMENT**). Increasing a progress indicator's value fills the control's gauge from left to right.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the progress indicator is used in. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the dialog box identifier is required.
Identifier	Lets you specify a string variable that identifies the progress indicator. This parameter is optional if the function is used in a **WITH...END WITH** construct. If this parameter is used, the period (**.**) after the identifier is required.

Note

- If a step causes a progress indicator to surpass its maximum value (**SETMAXRANGE**), the progress indicator value is reset to its minimum value (**SETMINRANGE**).

{button ,AL(`setincrement;getincrement;;;;',0,"Defaultoverview",)} **Related Topics**

{button ,AL(`progress_dyn;;;;;',0,"Defaultoverview",)} **Used with the following controls**

WITH...END WITH statements

WITH { DialogID.Identifier | DialogID | **.**Identifier }
    [statements]
END WITH

The **WITH...END WITH** construct enables you to execute dynamic dialog box functions without having to refer to a dynamic dialog box or dynamic dialog box controls. For example, if you have to refer to the same dialog box for many function calls, using the **WITH...END WITH** construct means you only have to refer to the dialog box once.

Parameter	Description
DialogID	Lets you specify a string variable that identifies the dynamic dialog box that the dynamic dialog box functions refer to.
Identifier	Lets you specify a string variable that identifies the dynamic dialog box control that the dynamic dialog box functions refer to. If this parameter is used, the period (**.**) before the identifier is required.
[statements]	Series of script instructions to execute. Dynamic dialog box functions that don't use dialog or control identifiers refer to dynamic dialog box or dynamic dialog box controls specified in the preceding **WITH** statement.

Note

- You can nest **WITH...END WITH** statements inside each other.

Example

The following three examples all perform the same actions:

```
'Example 1 (no WITH reference to dialog box or controls)
    MyDialog.Checkbox1.SETTEXT "Hello"
    MyDialog.CheckBox1.SETVALUE 1
    MyDialog.Checkbox1.SETTHREESTATE 0

'Example 2 (WITH reference to dialog box)
WITH MyDialog
    .Checkbox1.SETTEXT "Hello"
    .CheckBox1.SETVALUE 1
    .Checkbox1.SETTHREESTATE 0
END WITH

'Example 3 (WITH reference to dialog box and control)
WITH MyDialog.Checkbox1
    .SETTEXT "Hello"
    .SETVALUE 1
    .SETTHREESTATE 0
END WITH
```

{button ,AL(`ole_cs;end;corel_script_programming_language;Executing_script_files;Corel_SCRIPT_advanced_programming_features;;',0,"Defaultoverview",)} Related Topics

Error codes

If errors occur when you execute a script, an error message is displayed in the Compiler Output window. If you're using an error-handling routine, the Corel SCRIPT global variable ERRNUM is assigned a value based on the error. For more information about ERRNUM, see the **ONERROR** statement.

ERRNUM Value	Error Message
1	Internal Error
2	Unrecoverable error
3	Variable is not initialized
50	Division by 0
51	Type mismatch
52	Out of memory
53	Overflow
54	Invalid array index
55	Invalid specification of array dimensions
56	Resume without error
57	Out of resources
58	Stack Overflow
59	RETURN without GOSUB
200	Invalid file number
201	Unable to open file
202	File handle out of range
203	Error writing to file
204	Error reading file
205	Invalid file position
206	Cannot close file
250	Unable to change folder
251	Unable to make folder
252	Unable to remove folder
253	Unable to delete file
300	Incorrect parameter count
301	Unable to get DLL entry point
302	Unable to execute function
303	Object not initialized through WITHOBJECT
304	Invalid parameter
305	Undefined parameter
350	Unable to initialize dialog
351	Invalid dialog
352	More than one instance
353	Event handler misuse
600	OLE Automation Error
601	Invalid number of parameters

602	Bad variable type
603	Object does not support this function
604	Object does not support named arguments
605	Value out of range
606	Invalid Parameter name
607	Argument Type mismatch
608	Unknown Interface
609	Unknown language
610	Unspecified parameter is not optional
611	Could not create object
800 to 999	User-defined errors numbers.
1000 +	Corel application errors

Note

- Not all Corel SCRIPT errors are trappable.
- The **AppResponseTimeout** setting in the [Settings] section of the SCEDIT.INI file sets the default timeout for OLE Server commands. This setting is expressed in seconds. If you often receive Server Busy warnings you may want to increase this setting (the default is 45). The SCEDIT.INI file can be edited with any text editor and is located in C:\COREL\GRAPHICS8\SCRIPTS folder. You should not have Corel SCRIPT Editor running when editing SCEDIT.INI.

{button ,AL(`script_errors;;;;;',0,"Defaultoverview",)} Related Topics

Trappable Error: 1

ERRNUM 1 indicates an internal error.

Possible causes:

- Your system resources may be low.
- You may be low on hard disk space.

Possible solutions:

- Free up system resources and hard disk space.

Trappable Error: 2

ERRNUM 2 indicates an unrecoverable error. In Corel SCRIPT 6, this error was called unknown.

Possible causes:

- Your system resources may be low.
- You may be low on hard disk space.

Possible solutions:

- Free up system resources and hard disk space.

Trappable Error: 3

ERRNUM 3 indicates an uninitialized variable. This error number is only available in Corel SCRIPT 6.

Possible causes:

- You may have misspelled a variable name.
- You may not have initialized the variable.

Possible solutions:

- Confirm the spelling of the variables in your script.
- Confirm that all variables are initialized.

Trappable Error: 50

ERRNUM 50 indicates an attempt to divide by zero.

Possible causes:

- You may have misspelled a variable being used as a denominator.

Possible solutions:

- Ensure that all variables are spelled correctly.

Trappable Error: 51

ERRNUM 51 indicates a type mismatch.

Possible causes:

- You may be using the wrong operator.
- You may be mis-assigning a value to a variable.
- You may have misspelled the variable name.

Possible solutions:

- Ensure you are using the correct operators.
- Confirm that all variables are spelled correctly.
- Confirm that you are assigning values to the correct variables.

Trappable Error: 52

ERRNUM 52 indicates that the system is out of memory.

Possible causes:

- You may have an infinite loop in your script.
- You may have an over-sized array.

Possible solutions:

- Check the logic in your script, ensuring that infinite loops are removed.
- Ensure that all arrays are assigned a reasonable size.

Trappable Error: 53

ERRNUM 53 indicates an overflow.

Possible causes:

- You may have assigned an inappropriate value to a variable type, such as assigning a number greater than 32,767 to an integer.

Possible solutions:

- Trace through the script's logic, ensuring that all values are appropriate.
- Confirm your variable declarations to ensure than each variable is of the correct data type.

Trappable Error: 54

ERRNUM 54 indicates an invalid array index.

Possible causes:

- You may have tried to assign a value to an array element that does not exist. Use the REDIM statement to re-dimension the array.
- You may not have correctly initialized your index or your array.
- Your logic might be incorrectly adding to your index.

Possible solutions:

- Ensure that your array and index are initialized.
- Check the logic in your script, confirming all modifications to the array index.

Trappable Error: 55

ERRNUM 55 indicates an invalid specification of array dimensions.

Possible causes:

- The number of dimensions doesn't match the definition in the DIM statement.
- Invalid specification of an array dimension such as 7 to -3.
- Invalid specification of the dimension in the LBOUND or UBOUND statement.

Possible solutions:

- Correct the **DIM**, **LBOUND**, or **UBOUND** statement.

Trappable Error: 56

ERRNUM 56 indicates a RESUME statement encountered without error occurring.

Possible causes:

- Your script may have executed a **RESUME** statement outside of an active error-handling routine.

Possible solutions:

- You should place a **STOP**, **EXIT FUNCTION**, or an **EXIT SUB** before an error-handling routine in a script to prevent it from being executed when no error has occurred.

Trappable Error: 57

ERRNUM 57 indicates an out-of-resource error.

Possible causes:

- Too many Windows applications that are opened are resource intensive.

Possible solutions:

- Close some applications. If the error persists, restart Windows.

Trappable Error: 58

ERRNUM 58 indicates the stack has reached an abnormal size. This error is a warning that can be ignored.

Possible causes:

- Too many **GOSUB** calls
- Too many function calls.

Possible solutions:

- Check for a missing **RETURN** statement that should be used with a **GOSUB** statement.

Trappable Error: 59

ERRNUM 59 indicates the **RETURN** statement does not have a matching **GOSUB** statement.

Possible causes:

- Missing **GOSUB** statement.

Possible solutions:

- Check the script logic.

Trappable Error: 200

ERRNUM 200 indicates an invalid file number.

Possible causes:

- You may be assigning a file number outside the range of one to ten.
- You may be referring to a file that is not open.

Possible solutions:

- Confirm that your file numbers do not exceed ten. The **FREEFILE** function can help you find an unused file handle.

Trappable Error: 201

ERRNUM 201 indicates an inability to open a file.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may be corrupted.
- The file may not be of the correct file-type.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is both present and of the correct file-type.
- Confirm that the file is not corrupted.
- Confirm that the file is not already in use.

Trappable Error: 202

ERRNUM 202 indicates an out-of-range file handle. This error number is only available in Corel SCRIPT 6.0. See Error 200.

Possible causes:

- You may be assigning a file number outside the range of one to ten.

Possible solutions:

- Confirm that your file number is between one and ten. **FREEFILE** can help you find an unused file handle.

Trappable Error: 203

ERRNUM 203 indicates an error writing to a file.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may be corrupted.
- The file may not be of the correct file-type.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is both present and of the correct file-type.
- Confirm that the file is not corrupted, and is not already in use.

Trappable Error: 204

ERRNUM 204 indicates an error reading a file.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may be corrupted.
- The file may not be of the correct file-type.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is both present and of the correct file-type.
- Confirm that the file is not corrupted.
- Confirm that the file is not already in use.

Trappable Error: 205

ERRNUM 205 indicates an invalid file position.

Possible causes:

- You are trying to access an area outside the file.

Possible solutions:

- Use the **LOF** function to determine the file length, and access a position within that area.

Trappable Error: 206

ERRNUM 206 indicates a file cannot be closed.

Possible causes:

- File has not been opened.
- File may no longer exist.

Possible solutions:

- Check the logic in your script, confirming that the file has been opened, or that the file exists.

Trappable Error: 250

ERRNUM 250 indicates an error in changing files. This error number is only available in Corel SCRIPT 6.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may be corrupted.
- The file may not be of the correct file-type.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is both present and of the correct file-type.
- Confirm that the file is not corrupted, and is not already in use.

Trappable Error: 251

ERRNUM 251 indicates an error in creating a folder. This error number is only available in Corel SCRIPT 6.

Possible causes:

- You may not have file-creation privileges.
- The filename may already be present.

Possible solutions:

- Confirm that you are allowed to create files.
- Confirm that the file is not already in use.

Trappable Error: 252

ERRNUM 252 indicates an error in removing a folder. This error number is only available in Corel SCRIPT 6.

Possible causes:

- You may not be allowed to remove that file.
- The file may not be present.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is present.
- Confirm that the file is not already in use.

Trappable Error: 253

ERRNUM 253 indicates an error in deleting a file. This error number is only available in Corel SCRIPT 6.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is present.
- Confirm that the file is not already in use.

Trappable Error: 300

ERRNUM 300 indicates an incorrect parameter count. This error number is only available in Corel SCRIPT 6.

Possible causes:

- You may have omitted a parameter, or provided an unnecessary argument.

Possible solutions:

- Confirm that your parameter count is correct throughout the script.

Trappable Error: 301

ERRNUM 301 indicates a problem in getting a Dynamic Linked Library entry point. This error number is only available in Corel SCRIPT 6.

Possible causes:

- The DLL does not exist.
- The function does not exist in the DLL.
- You may have misspelled the function name, or the case may not match.

Possible solutions:

- Check that the DLL exists, and that the function is applicable to that DLL.
- Confirm the spelling and case of the function name.

Trappable Error: 302

ERRNUM 302 indicates an error in executing a function.

Possible causes:

- Your system resources may be low.
- You may be low on hard disk space.

Possible solutions:

- Free up system resources and hard disk space.

Trappable Error: 303

ERRNUM 303 indicates an object which hasn't been initialized through **WITHOBJECT**.

Possible causes:

- The **WITHOBJECT** statement may have been by-passed by flow-control statements, such as **GOTO**.

Possible solutions:

- Step through the flow of the script, to confirm execution of the **WITHOBJECT** statement.

Trappable Error: 304

ERRNUM 304 indicates an invalid parameter. This error number is only available in Corel SCRIPT 6.

Possible causes:

- You may have used parameter that is out of range.

Possible solutions:

- Confirm that the value of the parameters is in the acceptable range.
- Confirm data type of the parameters.

Trappable Error: 305

ERRNUM 305 indicates an undefined parameter.

Possible causes:

- Corel SCRIPT programming statement may use a parameter that is out of range.

Possible solutions:

- Confirm parameter value ranges.
- Confirm the script logic in determining the parameter values.

Trappable Error: 350

ERRNUM 350 indicates an error in initializing a dialog.

Possible causes:

- Your system resources may be low.
- You may be low on hard disk space.

Possible solutions:

- Free up system resources and hard disk space.

Trappable Error: 351

ERRNUM 351 indicates an invalid dialog. This error number is only available in Corel SCRIPT 6.

Possible causes:

- Your dialog may contain invalid controls, such as an incorrectly placed button.

Possible solutions:

- Confirm that the dialog controls are positioned within reasonable bounds.

Trappable Error: 352

ERRNUM 352 indicates that the script attempted to run more than one instance of a custom dialog box at the same time.

Possible causes:

- A **DIALOG** statement called custom dialog box that was already open.

Possible solutions:

- Check the logic in your script, confirming the location of the **DIALOG** statements.

Trappable Error: 353

ERRNUM 353 indicates that a Corel SCRIPT statement or function that can only be used in the dialog event handler subroutine, was used outside of it. Click ⇨ for more information.

Possible causes:

- A missing or improperly defined subroutine.

Possible solutions:

- Check the script logic.

Trappable Error: 600

ERRNUM 600 indicates an error in OLE automation.

Possible causes:

- The application may have problems opening.
- The object name may be incorrect.

Possible solutions:

- Check the OLE automation errors.

Trappable Error: 601

ERRNUM 601 indicates an invalid number of parameters.

Possible causes:

- The application is expecting a different number of parameters.

Possible solutions:

- Check that all parameters are required, and that all required parameters are there.

Trappable Error: 602

ERRNUM 602 indicates a bad variable type.

Possible causes:

- You have passed the function a parameter of an unknown variable type.

Possible solutions:

- Confirm the variable's data type.

Trappable Error: 603

ERRNUM 603 indicates that the object doesn't support this function.

Possible causes:

- The object doesn't support this function.

Possible solutions:

- Confirm that this function is supported by the object.

Trappable Error: 604

ERRNUM 604 indicates that the object doesn't support the named arguments.

Possible causes:

- Some applications do not support named arguments.

Possible solutions:

- Confirm that the arguments are suitable for the application.

Trappable Error: 605

ERRNUM 605 indicates a value out of the acceptable range.

Possible causes:

- You may have passed a large value to an application which expected a smaller value.

Possible solutions:

- Confirm that the value you are trying to pass to the application is within an acceptable range, and is of a suitable data type.

Trappable Error: 606

ERRNUM 606 indicates an invalid parameter name.

Possible causes:

- You may have passed an incorrectly spelled named argument.

Possible solutions:

- Confirm that all named arguments are recognized.

Trappable Error: 607

ERRNUM 607 indicates an argument type mismatch.

Possible causes:

- You may have passed a variable which is not of the appropriate data type.

Possible solutions:

- Confirm that the data type is appropriate.

Trappable Error: 608

ERRNUM 608 indicates an unknown interface.

Possible causes:

- The object does not support OLE automation.

Possible solutions:

- Change the object, if necessary.

Trappable Error: 609

ERRNUM 609 indicates an unknown language.

Possible causes:

- The application is unable to recognize the international language of the functions.

Possible solutions:

- Confirm that the language is compatible.

Trappable Error: 610

ERRNUM 610 indicates that a required parameter is missing.

Possible causes:

- A required parameter is missing.

Possible solutions:

- Confirm that all required parameters are present.

Trappable Error: 611

ERRNUM 611 indicates that an OLE Automation object could not be created.

Possible causes:

- Error specifying the object name in the **WITHOBJECT** statement.

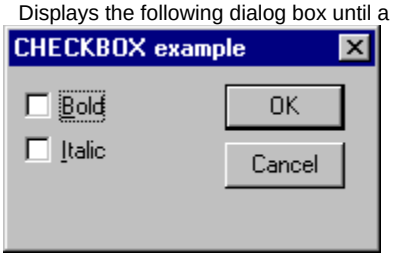
Possible solutions:

- Check for a missing **WITHOBJECT** statement.
- Check for a spelling mistake in a **WITHOBJECT** statement.

Check box example (static dialog)

```
' Check box example
BEGIN DIALOG Dialogbox1 122, 55, "CHECKBOX example"
    CHECKBOX 5, 7, 60, 10, "&Bold", bold%
    CHECKBOX 5, 20, 60, 10, "&Italic", ital%
    OKBUTTON 72, 5, 40, 14
    CANCELBUTTON 72, 23, 40, 14
END DIALOG
ret = DIALOG(Dialogbox1)
    ' If ret is 2, then Cancel button was chosen
IF ret = 2 THEN STOP

REM Displays a MESSAGE based on selections
IF bold=1 and ital=1 then
    MESSAGE "Bold on; Italic on"
ELSEIF bold=1 and ital=0 then
    MESSAGE "Bold on; Italic off"
ELSEIF bold=0 and ital=1 then
    MESSAGE "Bold off; Italic on"
ELSE
    MESSAGE "Bold off; Italic off"
ENDIF
```

Displays the following dialog box until a OK or CANCEL selected.



Note

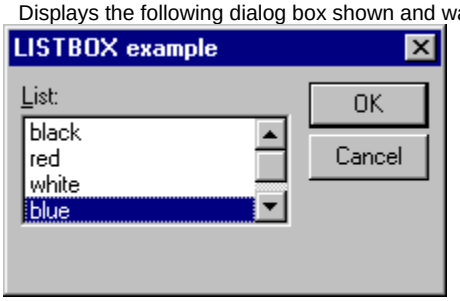
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

List box example (static dialog)

```
' List box example
REM Create an array called arr
DIM arr$(5)
arr(1) = "black"
arr(2) = "red"
arr(3) = "white"
arr(4) = "blue"
arr(5) = "green"
index = 4
BEGIN DIALOG listboxdlg 144, 68, "LISTBOX example"
    TEXT 4, 4, 90, 8, "&List:"
    LISTBOX 4, 14, 90, 50, arr, index
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(listboxdlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP
MESSAGE "You chose " + arr(index)
```

Displays the following dialog box shown and waits for the user to select an element in the list box and then choose OK.



Note

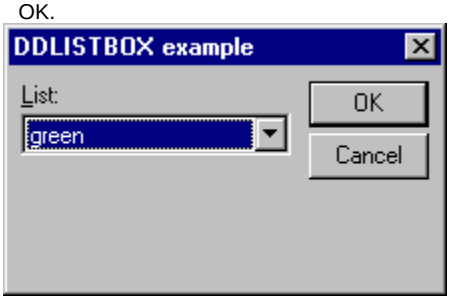
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Drop-down list box example (static dialog)

```
' Drop-down list box example
REM Create an array called arr
DIM arr$(5)
arr(1) = "black"
arr(2) = "red"
arr(3) = "white"
arr(4) = "blue"
arr(5) = "green"
index% = 4'"green"
BEGIN DIALOG DDlistboxdlg 144, 68, "DDLISTBOX example"
    TEXT 4, 4, 90, 8, "&List:"
    DDLISTBOX 4, 14, 90, 50, arr, index
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(DDlistboxdlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP
MESSAGE "You chose " + arr(index)
```

Displays the following dialog box shown and waits for the user to select an element in the drop-down list box and then choose OK.



Note

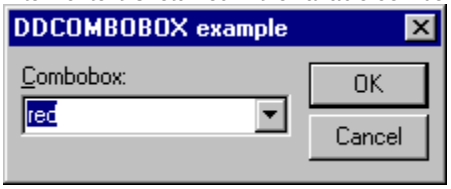
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Drop-down combo box example (static dialog)

```
' Drop-down combo box example
REM Create an array called arr
DIM arr$(5)
arr(1) = "black"
arr(2) = "red"
arr(3) = "white"
arr(4) = "blue"
arr(5) = "green"
combo = arr(2) 'initializes a choice in the combobox
BEGIN DIALOG ddcombodlg 144, 40, "DDCOMBOBOX example"
    TEXT 4, 4, 90, 8, "&Combobox:"
    DDCOMBOBOX 4, 14, 90, 62, arr, combo
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(ddcombodlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP

FOR i% = 1 TO 5

' If we are within the bounds of the array and
    ' find a match, then display a message.
    IF combo = arr(i) THEN
        MESSAGE "You chose " + combo
        GOTO ENDFOR
    ENDIF
NEXT i
REM If a match is found, this line will be skipped.
MESSAGE "You chose your own color: " + combo
ENDFOR:
```

Displays the dialog box until the user selects an item from the list or enters text into the text box and selects OK. The selected item or text is returned in the variable **combo**.



Note

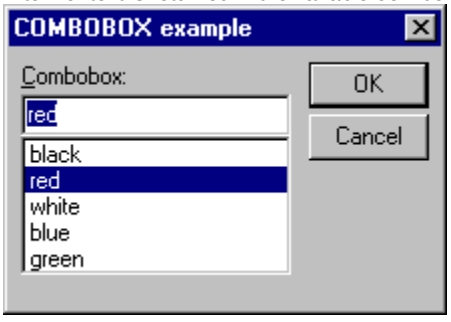
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Combo box example (static dialog)

```
' Combo box example
REM Create an array called arr
DIM arr$(5)
arr(1) = "black"
arr(2) = "red"
arr(3) = "white"
arr(4) = "blue"
arr(5) = "green"
combo = arr(2) 'initializes a choice in the combobox
BEGIN DIALOG combodlg 144, 80, "COMBOBOX example"
    TEXT 4, 4, 90, 8, "&Combobox:"
    COMBOBOX 4, 14, 90, 62, arr, combo
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(combodlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP

FOR i% = 1 TO 5

' If we are within the bounds of the array and
    ' find a match, then display a message.
    IF combo = arr(i) THEN
        MESSAGE "You chose " + combo
        STOP
    ENDIF
NEXT i
REM If a match is found, this line will be skipped.
MESSAGE "You chose your own color: " + combo
```

Displays the dialog box until the user selects an item from the list or enters text into the text box and selects OK. The selected item or text is returned in the variable **combo**.



Note

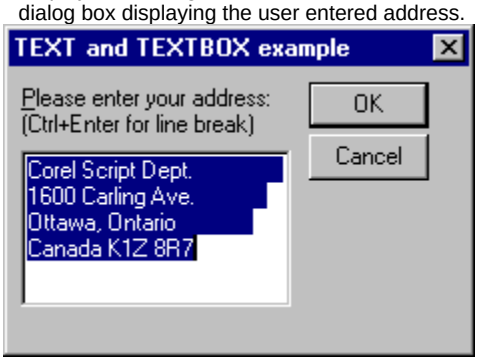
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Text and text box example (static dialog)

```
' Text and text box example
text1 = "Corel Script Dept."+CHR(13)+CHR(10)+"1600 Carling Ave."+CHR(13)+CHR(10)+"Ottawa,
Ontario"+CHR(13)+CHR(10)+"Canada K1Z 8R7"
label1 = "&Please enter your address:" + CHR(13) + "(CTRL+ENTER for line break)"
' Initialize dialog box variables
'index = 2
BEGIN DIALOG textdlg 153, 87, "TEXT and TEXTBOX example"
    TEXT  4, 5, 100, 16, label1
    TEXTBOX 4, 25, 90, 48, text1
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(textdlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP
MESSAGE "Your Address is: " + CHR(13) + text1
```

Displays the dialog box shown below. In the text box, use CTRL+ENTER for line breaks. The dialog box returns a message dialog box displaying the user entered address.



Note

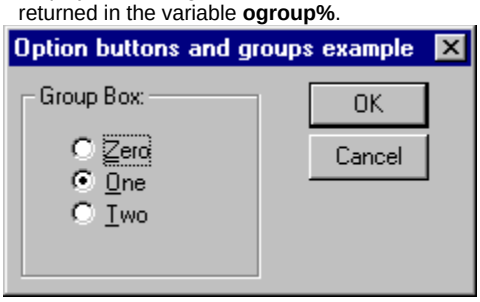
- If you want to insert a line break into the default text for a text box, use ANSI characters 10 and 13. See **CHR** for more information.
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Option button, option group, and group box example (static dialog)

```
' Option button, option group, and group box example
ogroup%=1  'sets the default option button to 1
BEGIN DIALOG OptionDialog 154, 68, "Option buttons and groups example"
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
    GROUPBOX 4, 4, 80, 60, "Group Box:"
    OPTIONGROUP ogroup%
        OPTIONBUTTON 20, 20, 30, 10, "&Zero"
        OPTIONBUTTON 20, 30, 30, 10, "&One"
        OPTIONBUTTON 20, 40, 30, 10, "&Two"
END DIALOG
ret_val = DIALOG(Optiondialog)
' If ret_val is 2, then Cancel button was chosen
IF ret_val = 2 THEN STOP
SELECT CASE ogroup
    CASE 0
        ' The Zero option button was selected
        MESSAGE "You chose Zero"
    CASE 1
        ' The One option button was selected
        MESSAGE "You chose One"
    CASE 2
        ' The Two option button was selected
        MESSAGE "You chose Two"
END SELECT
```

Displays the dialog box shown below until the user selects one of the options buttons and OK. The response value is then returned in the variable **ogroup%**.



Note

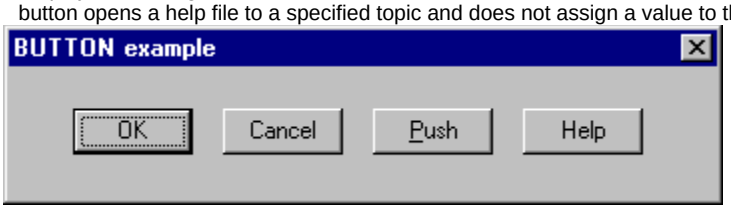
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

OK, Cancel, Help, and Push button example (static dialog)

```
' Push button, OK button, Cancel button, and Help button example
BEGIN DIALOG Buttons1 55, 34, 236, 40, "BUTTON example"
    OKBUTTON 21, 12, 40, 14
    CANCELBUTTON 71, 12, 40, 14
    PUSHBUTTON 121, 12, 40, 14, "&Push"
    HELPBUTTON 171, 12, 40, 14, "C:\yourhelp\help.hlp", 104
END DIALOG
ret = DIALOG(Buttons1)
' If ret is 1, then OK button was chosen
IF ret = 1 THEN MESSAGE "OK button chosen"
' If ret is 2, then Cancel button was chosen
IF ret = 2 THEN
    MESSAGE "CANCEL button was chosen"
    STOP
END IF
' If ret is 3, then Push button was chosen
IF ret = 3 THEN MESSAGE "Push button chosen"
```

Displays the dialog box shown below until the user selects either the OK button, Cancel Button, or the Push button. The help button opens a help file to a specified topic and does not assign a value to the **ret** variable



Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

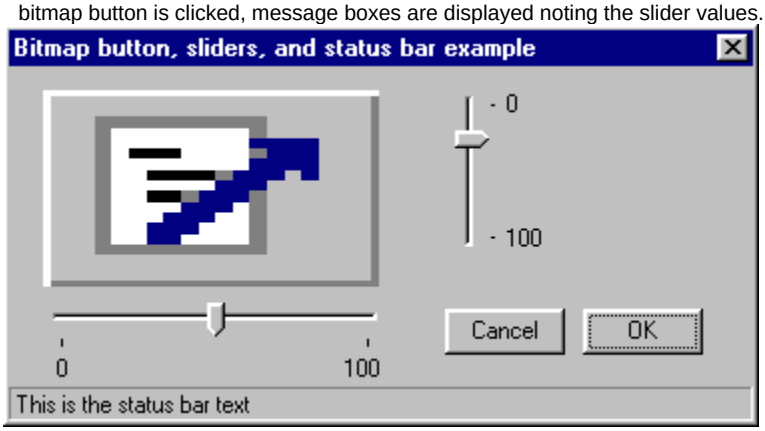
{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Bitmap button, sliders, and status bar example (static dialog)

```
' Bitmap button, sliders, and status bar example

'Set array for the bitmap button
GLOBAL BitmapButton1Arr(3) as string
BitmapButton1Arr(1) = "C:\WINDOWS\NORMAL.bmp"
BitmapButton1Arr(2) = "C:\WINDOWS\DEPRESSED.bmp"
BitmapButton1Arr(3) = "C:\WINDOWS\FOCUS.bmp"

'Sets initial value for the vertical and horizontal slider
VSlider1Val% = 25
HSlider1Val% = 50

'Sets status bar text
Status1Txt$ = "This is the status bar text"

BEGIN DIALOG Script1 248, 108, "Bitmap button, sliders, and status bar example"
    OKBUTTON  191, 74, 40, 14
    CANCELBUTTON  145, 74, 40, 14
    BITMAPBUTTON  11, 7, 112, 60, BitmapButton1Arr$
    HSLIDER 10, 71, 117, 16, HSlider1Val%
    VSLIDER 147, 4, 15, 55, VSlider1Val%
    STATUS Status1Txt$
    TEXT  15, 88, 23, 8, "0"
    TEXT  165, 7, 23, 8, "0"
    TEXT  165, 48, 23, 8, "100"
    TEXT  111, 88, 13, 8, "100"
END DIALOG
ret = DIALOG(Script1)

' If ret is 1, then OK button was chosen
IF ret = 1 THEN
    MESSAGE "OK button chosen"
    MESSAGE "Horizontal slider: " & HSlider1Val%
    MESSAGE "Vertical slider: " & VSlider1Val%
ENDIF

' If ret is 2, then Cancel button was chosen
IF ret = 2 THEN
    MESSAGE "CANCEL button was chosen"
    STOP
END IF

' If ret is 3, then Bitmap button was chosen
IF ret = 3 THEN
    MESSAGE "Bitmap button chosen"
    MESSAGE "Horizontal slider: " & HSlider1Val%
    MESSAGE "Vertical slider: " & VSlider1Val%
ENDIF
```

Displays the dialog box shown below until the user selects either the OK button, Cancel Button, or the bitmap button. If the OK or

bitmap button is clicked, message boxes are displayed noting the slider values.



Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Example of all dialog controls (static dialog)

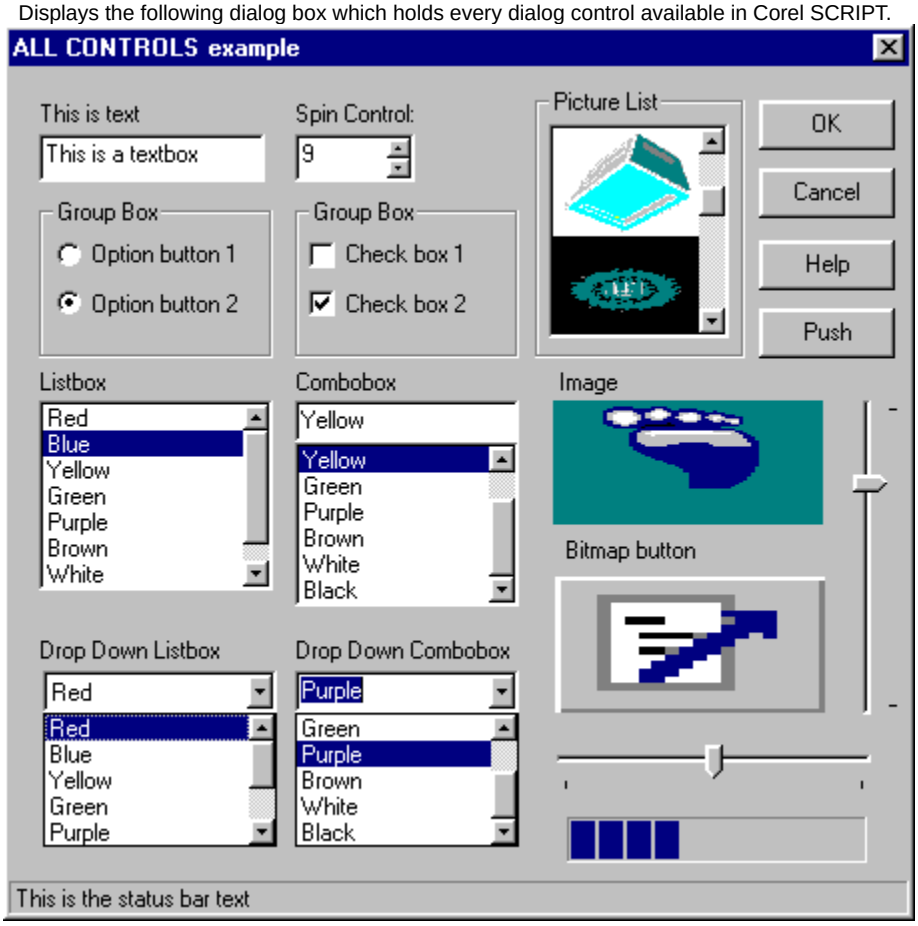
' Example showing every dialog control used in Corel SCRIPT

```
'Set array for image list
DIM bmparray$(5)
bmpname = "d:\corel60\photopnt\plgbrush\footprnt.bmp"
bmparray(1) = "d:\corel60\photopnt\plgbrush\fuzzy.bmp"
bmparray(2) = "d:\corel60\photopnt\plgbrush\boxpanel.bmp"
bmparray(3) = "d:\corel60\photopnt\plgbrush\textart.bmp"
bmparray(4) = "d:\corel60\photopnt\plgbrush\treebare.bmp"
bmparray(5) = "d:\corel60\photopnt\plgbrush\saturn.bmp"

'Set array for list boxes
DIM color(8) as string
color(1) = "Red"
color(2) = "Blue"
color(3) = "Yellow"
color(4) = "Green"
color(5) = "Purple"
color(6) = "Brown"
color(7) = "White"
color(8) = "Black"

'Set array for the bitmap button
DIM BitmapButton1Arr(3) as string
BitmapButton1Arr(1) = "C:\WINDOWS\NORMAL.bmp"
BitmapButton1Arr(2) = "C:\WINDOWS\DEPRESSED.bmp"
BitmapButton1Arr(3) = "C:\WINDOWS\FOCUS.bmp"

'initial values for other dialog controls
text1 = "This is text"
help = "HELP!"
push1 = "Push button"
Progress1Val% = 33

'Sets initial value for the vertical and horizontal slider
VSlider1Val% = 25
HSlider1Val% = 50

'Sets status bar text
Status1Txt$ = "This is the status bar text"

'Dialog declaration
BEGIN DIALOG d1 300, 260, "ALL CONTROLS example"
    OKBUTTON  250, 10, 45, 15
    CANCELBUTTON  250, 31, 45, 15
    HELPBUTTON  250, 53, 45, 15, help, HelpID%
    PUSHBUTTON  250, 74, 45, 15, "Push"
    TEXT  10, 10, 80, 9, "This is text"
    TEXTBOX  10, 20, 75, 15, textbox1$
```

```
    GROUPBOX  10, 40, 78, 49, "Group Box"
    OPTIONGROUP group1%
        OPTIONBUTTON  15, 50, 66, 15, "Option button 1"
        OPTIONBUTTON  15, 65, 66, 15, "Option button 2"
    GROUPBOX  95, 40, 75, 49, "Group Box"
    CHECKBOX  100, 50, 65, 15, "Check box 1", binary1%
    CHECKBOX  100, 65, 65, 15, "Check box 2", binary2%
    TEXT  10, 92, 70, 15, "Listbox"
    LISTBOX  10, 102, 78, 70, Color, listbox1%
    TEXT  95, 92, 70, 10, "Combobox"
    COMBOBOX  95, 102, 75, 70, Color, combobox1$
    TEXT  10, 175, 70, 15, "Drop Down Listbox"
    DDLISTBOX  10, 185, 78, 59, Color, ddlistbox1%
    TEXT  95, 175, 75, 15, "Drop Down Combobox"
    DDCOMBOBOX  95, 185, 75, 59, Color, ddcombobox1$
    TEXT  95, 10, 80, 15, "Spin Control: "
    SPINCONTROL  95, 20, 40, 15, spin1%
    IMAGE  181, 102, 90, 38, Bmpname
    GROUPBOX  175, 6, 70, 83, "Picture List"
    IMAGELISTBOX  180, 17, 60, 66, bmparray, imglst%
    TEXT  183, 92, 50, 8, "Image"
    BITMAPBUTTON  182, 156, 90, 42, BitmapButton1Arr$
    TEXT  185, 144, 50, 8, "Bitmap button"
    VSLIDER 280, 98, 16, 105, VSlider1Val%
    PROGRESS 186, 230, 100, 14, Progress1Val%
    HSLIDER 178, 207, 114, 16, HSlider1Val%
    STATUS Status1Txt$
END DIALOG

'Other dialog control defaults
textbox1 = "This is a textbox"
group1 = 1
binary1% = -1
binary2% = 1
binary3% = 0
Listbox1 = 2
Combobox1 = color(3)
DDListBox1 = 1
DDCombobox1 = color(5)
spin1 = 9
imglst = 2

TRYAGAIN:
ret = DIALOG(d1) 'displays dialog box
IF ret = 2 THEN STOP

IF ret = 3 THEN
    MESSAGE "You pressed a push button"
    GOTO TRYAGAIN
ENDIF
```

```
IF ret = 4 THEN
    MESSAGE "You pressed a bitmap button"
    GOTO TRYAGAIN
ENDIF

IF ret = 1 THEN
'message box text
mess$ = "You entered " + CHR(34) + textbox1 + CHR(34) + " in the textbox"
mess = mess + CHR(13) + "You picked " + CSTR(spin1) + " in the spin box"
mess$ = mess + CHR(13) + "You selected Option Button #" + CSTR(group1 + 1)
mess = mess + CHR(13) + "You selected:"
if binary1<>0 then mess = mess + " Check Button #1"
if binary2<>0 then mess = mess + " Check Button #2"
mess = mess + CHR(13) + "You picked " + color(Listbox1) + " in the ListBox"
mess = mess + CHR(13) + "You entered " + Combobox1 + " in the ComboBox"
mess = mess + CHR(13) + "You entered " + DDCombobox1 + " in the DDComboBox"
mess = mess + CHR(13) + "You picked " + color(DDListbox1) + " in the DDListBox"
mess = mess + CHR(13) + "You picked " + bmparray(imglst) + " as your image"
mess = mess + CHR(13) + "You set the horizontal slider to " & HSlider1Val%
mess = mess + CHR(13) + "You set the vertical slider to " & VSlider1Val%
MESSAGE mess
ENDIF
```

Displays the following dialog box which holds every dialog control available in Corel SCRIPT.



Note

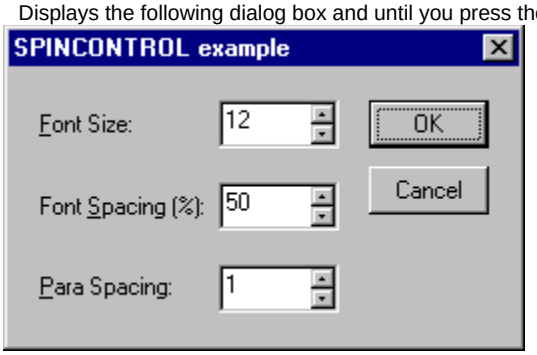
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} **Related Topics**

Spin control example (static dialog)

```
' Spin control example
REM set default values
spin1% = 12
spin2% = 50
spin3% = 1

BEGIN DIALOG spindlg 170, 85, "SPINCONTROL example"
    TEXT 10, 13, 55, 15, "&Font Size: "
        SPINCONTROL 70, 10, 40, 15, spin1%
    TEXT 10, 38, 55, 15, "Font &Spacing (%): "
        SPINCONTROL 70, 35, 40, 15, spin2%
    TEXT 10, 63, 55, 15, "&Para Spacing: "
        SPINCONTROL 70, 60, 40, 15, spin3%
    OKBUTTON 120, 10, 40, 15
    CANCELBUTTON 120, 30, 40, 15
END DIALOG

return = DIALOG(spindlg)
IF return = 2 THEN STOP

MESSAGE "You chose a font size of: " + CSTR(spin1)
MESSAGE "You chose a font spacing of: " + CSTR(spin2) + "%"
MESSAGE "You chose a para spacing of: " + CSTR(spin3)
```

Displays the following dialog box and until you press the OK or Cancel button.



Note

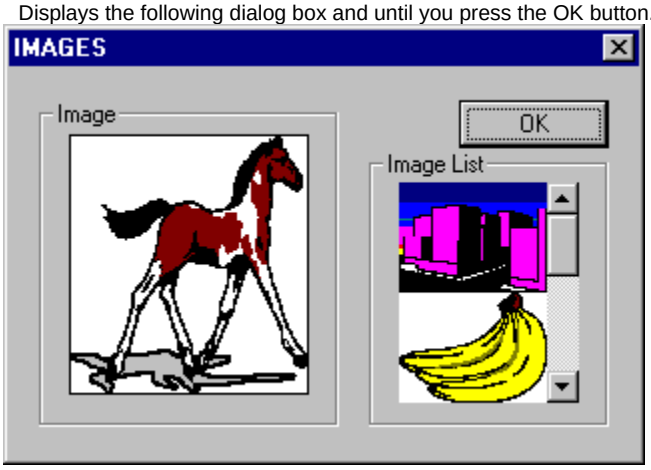
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} **Related Topics**

Image list box and Image example (static dialog)

```
' Image list box and Image example
DIM bmparray$(5)
bmpname = "C:\corel60\photopnt\plgbrush\pony.bmp"
bmparray(1) = "C:\corel60\photopnt\plgbrush\building.bmp"
bmparray(2) = "C:\corel60\photopnt\plgbrush\banana.bmp"
bmparray(3) = "C:\corel60\photopnt\plgbrush\textart.bmp"
bmparray(4) = "C:\corel60\photopnt\plgbrush\treebare.bmp"
bmparray(5) = "C:\corel60\photopnt\plgbrush\saturn.bmp"
BEGIN DIALOG D1 210, 120, "IMAGES"
    OKBUTTON 150, 10, 50, 15
    GROUPBOX 10, 10, 100, 100, "Image"
    IMAGE 20, 20, 80, 80, Bmpname
    GROUPBOX 120, 25, 80, 85, "Image List"
        IMAGELISTBOX 130, 35, 60, 100, bmparray, imglst%
END DIALOG
ret% = DIALOG(D1)
```

Displays the following dialog box and until you press the OK button.



Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Example 1: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 160, 118, "International Sales", SUB Sales
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  10, 10, 60, 10, .OptionButton1, "North America"
        OPTIONBUTTON  100, 10, 48, 10, .OptionButton2, "Europe"
    GROUPBOX  5, 3, 150, 20, .GroupBox1
    CHECKBOX  10, 35, 50, 10, .CheckBox1, "Canada"
    CHECKBOX  10, 55, 50, 10, .CheckBox2, "Mexico"
    CHECKBOX  10, 75, 50, 10, .CheckBox3, "USA"
    CHECKBOX  100, 35, 50, 10, .CheckBox4, "Ireland"
    CHECKBOX  100, 55, 50, 10, .CheckBox5, "Italy"
    CHECKBOX  100, 75, 50, 10, .CheckBox6, "Spain"
    OKBUTTON  115, 100, 40, 14, .OK1
    CANCELBUTTON  70, 100, 40, 14, .Cancel1
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB Sales(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set the value of the first
'button and to disable three check boxes
IF Event=0 THEN
    Dialog1.OptionButton1.SETVALUE 1
    Dialog1.CheckBox4.ENABLE FALSE
    Dialog1.CheckBox5.ENABLE FALSE
    Dialog1.CheckBox6.ENABLE FALSE
ENDIF

'Initialization condition to set the check boxes to 2-state
IF Event=0 THEN
    Dialog1.CheckBox1.SETTHREESTATE FALSE
    Dialog1.CheckBox2.SETTHREESTATE FALSE
    Dialog1.CheckBox3.SETTHREESTATE FALSE
    Dialog1.CheckBox4.SETTHREESTATE FALSE
    Dialog1.CheckBox5.SETTHREESTATE FALSE
    Dialog1.CheckBox6.SETTHREESTATE FALSE
ENDIF

'Enabling Europe enables 3 check boxes and
'disables the three others
IF ControlID=3 AND Event=2 THEN
    Dialog1.CheckBox1.ENABLE FALSE
    Dialog1.CheckBox2.ENABLE FALSE
    Dialog1.CheckBox3.ENABLE FALSE
    Dialog1.CheckBox4.ENABLE TRUE
    Dialog1.CheckBox5.ENABLE TRUE
    Dialog1.CheckBox6.ENABLE TRUE
ENDIF
```

```
'Enabling North America enables 3 check boxes and
'disables the three others
IF ControlID=2 AND Event=2 THEN
    Dialog1.CheckBox1.ENABLE TRUE
    Dialog1.CheckBox2.ENABLE TRUE
    Dialog1.CheckBox3.ENABLE TRUE
    Dialog1.CheckBox4.ENABLE FALSE
    Dialog1.CheckBox5.ENABLE FALSE
    Dialog1.CheckBox6.ENABLE FALSE
ENDIF

END SUB
```

The above script displays the following dialog boxes depending on the selected option button. Selecting an option button disables a set of check boxes.

Note

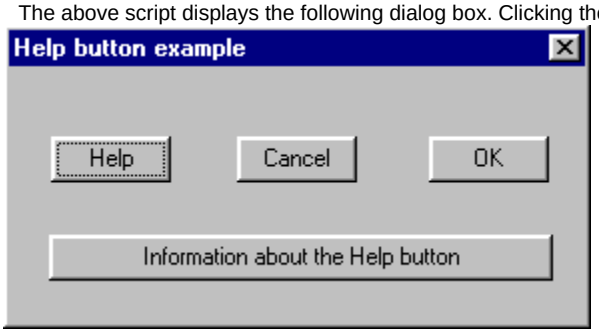
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 2: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 192, 78, "Help button example", SUB Buttons
    HELPBUTTON  14, 21, 40, 14, .Help1
    OKBUTTON  140, 21, 40, 14, .OK1
    CANCELBUTTON  76, 21, 40, 14, .Cancel1
    PUSHBUTTON  13, 51, 168, 14, .PushButton1, "Information about the Help button"
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB Buttons(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set the values for the option buttons
IF Event = 0 THEN
    Dialog1.Help1.SETHELPPATH("C:\WINDOWS\DESKTOP\MACRO\SCRIPT65\SCEDIT.HLP")
    Dialog1.Help1.SETHELPINDEX(1234567890)
ENDIF

'Displays message when push button pressed
IF ControlID = 4 AND Event = 2 THEN
    MESSAGE "The Help button opens " & Dialog1.Help1.GETHELPPATH ()
    MESSAGE "The topic " & Dialog1.Help1.GETHELPINDEX () & " is opened"
ENDIF

END SUB
```

The above script displays the following dialog box. Clicking the Help button opens a topic in the help file you're reading.



Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

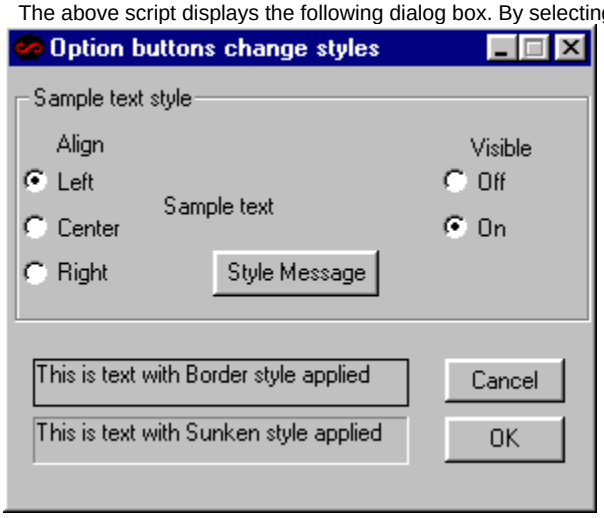
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

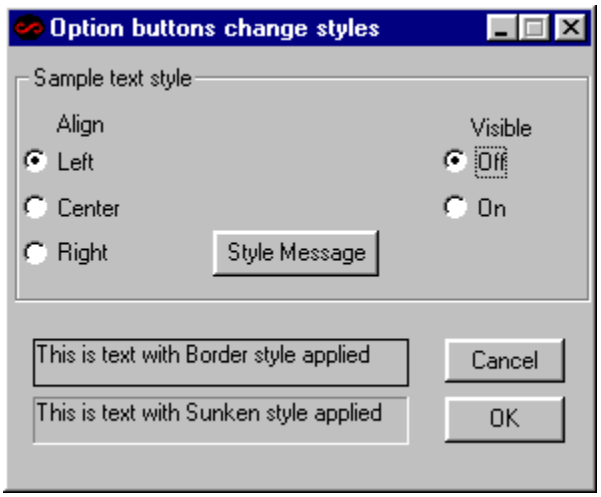
Example 3: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 194, 135, "Option buttons change styles", SUB Styles
    TEXT  51, 38, 80, 11, .Text1, "Sample text"
    GROUPBOX  2, 5, 192, 73, .GroupBox1, "Sample text style"
    TEXT  16, 19, 30, 8, .Text4, "Align"
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  4, 30, 43, 10, .OptionButton1, "Left"
        OPTIONBUTTON  4, 44, 48, 10, .OptionButton2, "Center"
        OPTIONBUTTON  4, 58, 48, 10, .OptionButton3, "Right"
    TEXT  154, 20, 30, 8, .Text5, "Visible"
    OPTIONGROUP .OptionGroup2
        OPTIONBUTTON  144, 30, 30, 10, .OptionButton4, "Off"
        OPTIONBUTTON  144, 44, 30, 10, .OptionButton5, "On"
    PUSHBUTTON  68, 56, 55, 14, .PushButton1, "Style Message"
    CANCELBUTTON  145, 89, 40, 14, .Cancel1
    OKBUTTON  145, 107, 40, 14, .OK1
    TEXT  8, 89, 125, 15, .Text2, "This is text with Border style applied"
    TEXT  8, 107, 125, 15, .Text3, "This is text with Sunken style applied"
END DIALOG

'Border, sunken, and title bar styles are applied outside the dialog event
'handler, and before the DIALOG statement
Dialog1.Text2.SETSTYLE 8
Dialog1.Text3.SETSTYLE 256
Dialog1.SETSTYLE 32
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB Styles(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set the values for the option buttons
IF Event=0 THEN
    Dialog1.OptionButton1.SETVALUE 1
    Dialog1.OptionButton5.SETVALUE 1
ENDIF

'changes the alignment setting
IF Event = 2 THEN
    IF Dialog1.OptionGroup1.GETVALUE() = 0 THEN
        Dialog1.Text1.SETSTYLE 16
    ELSEIF Dialog1.OptionGroup1.GETVALUE() = 1 THEN
        Dialog1.Text1.SETSTYLE 64
    ELSEIF Dialog1.OptionGroup1.GETVALUE() = 2 THEN
        Dialog1.Text1.SETSTYLE 32
    ENDIF
ENDIF

'changes the visible setting
IF Event = 2 THEN
```

```
    IF Dialog1.OptionGroup2.GETVALUE() = 0 THEN
        Dialog1.Text1.SETSTYLE 2
    ELSEIF Dialog1.OptionGroup2.GETVALUE() = 1 THEN
        Dialog1.Text1.SETSTYLE 1
    ENDIF
ENDIF

'Displays message box of current settings
IF ControlID = 12 AND Event = 2 THEN
    RV_Sample = Dialog1.Text1.GETSTYLE() 'retrieves settings

    'the following lines test for the alignment settings
    IF 64 AND RV_Sample THEN
        Align$ = "Center"
    ELSEIF 32 AND RV_Sample THEN
        Align$ = "Right"
    ELSE
        Align$ = "Left"
    ENDIF

    'the following lines test for the visible settings
    IF 1 AND RV_Sample THEN
        Vis$ = "Visible"
    ELSE
        Vis$ = "Invisible"
    ENDIF

    MESSAGE "The text is " & Align$ & " aligned" & CHR(13) & "and " & Vis$
ENDIF
END SUB
```

The above script displays the following dialog box. By selecting option buttons you change the style of the sample text.





Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

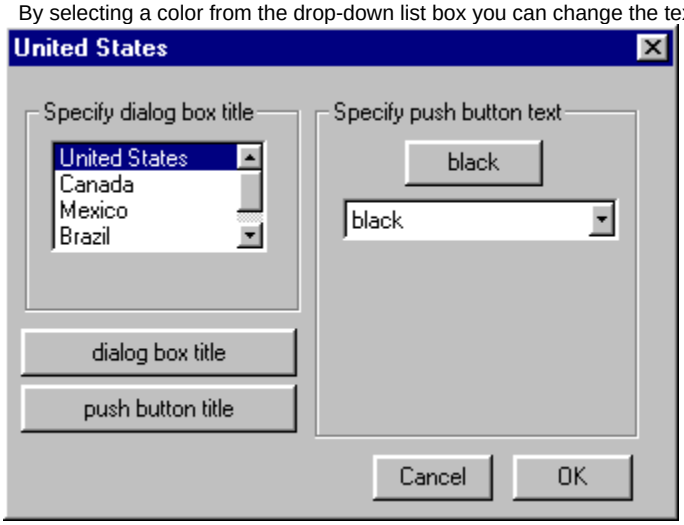
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

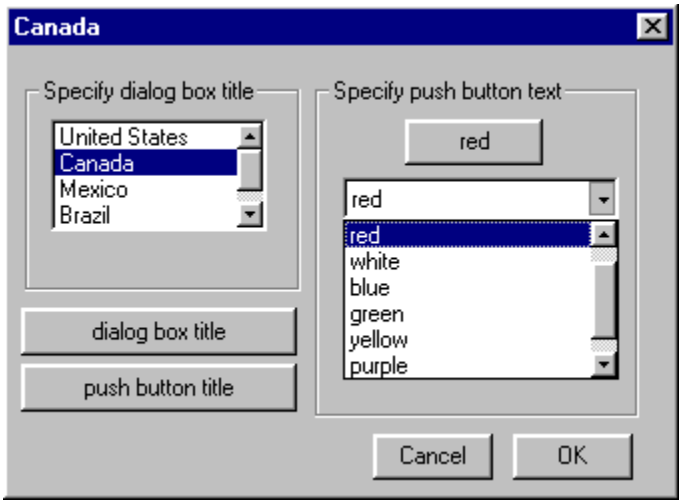
Example 4: Dynamic dialog box

```
'Set arrays for the list boxes
GLOBAL country$(5)
country(1) = "United States"
country(2) = "Canada"
country(3) = "Mexico"
country(4) = "Brazil"
country(5) = "Chile"
GLOBAL color$(8)
color(1) = "black"
color(2) = "red"
color(3) = "white"
color(4) = "blue"
color(5) = "green"
color(6) = "yellow"
color(7) = "purple"
color(8) = "brown"

'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 221, 137, "United States", SUB TextCond
    GROUPBOX  5, 9, 92, 66, .GroupBox1, "Specify dialog box title"
    GROUPBOX  102, 9, 108, 105, .GroupBox2, "Specify push button text"
    PUSHBUTTON  132, 22, 46, 14, .PushButton1, "black"
    LISTBOX  14, 22, 72, 46, .ListBox1
    DDLISTBOX  111, 40, 92, 68, .DDListBox2
    PUSHBUTTON  4, 80, 92, 15, .PushButton2, "dialog box title"
    PUSHBUTTON  4, 97, 92, 15, .PushButton3, "push button title"
    OKBUTTON  168, 119, 40, 14, .OK1
    CANCELBUTTON  121, 119, 40, 14, .Cancel1
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB TextCond(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set option button values
IF Event=0 THEN
    Dialog1.ListBox1.SETARRAY Country$
    Dialog1.DDListBox2.SETARRAY Color$
    'specifies the initial color selection
    Dialog1.DDListBox2.SETSELECT 1
    Dialog1.ListBox1.SETSELECT 1
ENDIF

'Country selection changes dialog box title
IF ControlID = 4 AND Event = 2 THEN
    Country_Num = Dialog1.ListBox1.GETSELECT()
    Dialog1.SETTEXT Country(Country_Num)
ENDIF
```

```
'Color selection changes push button text
IF ControlID = 5 AND Event = 2 THEN
    Color_Num = Dialog1.DDListBox2.GETSELECT()
    Dialog1.PushButton1.SETTEXT Color(Color_Num)
ENDIF

'Clicking push button displays dialog title text
IF ControlID = 6 AND Event = 2 THEN
    MESSAGE Dialog1.GETTEXT ()
ENDIF

'Clicking push button displays push button text
IF ControlID = 7 AND Event = 2 THEN
    MESSAGE Dialog1.PushButton1.GETTEXT ()
ENDIF

END SUB
```

The above script displays the following dialog box. By selecting a country from the list box you can change the dialog box title. By selecting a color from the drop-down list box you can change the text on the push button above the drop-down list box.





Clicking the other two push buttons open a message box displaying the dialog box title or the text on the color push button.

Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

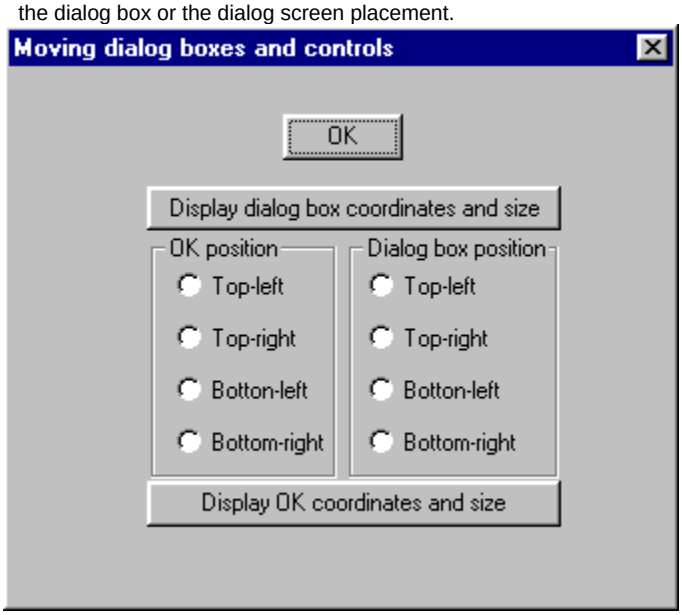
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

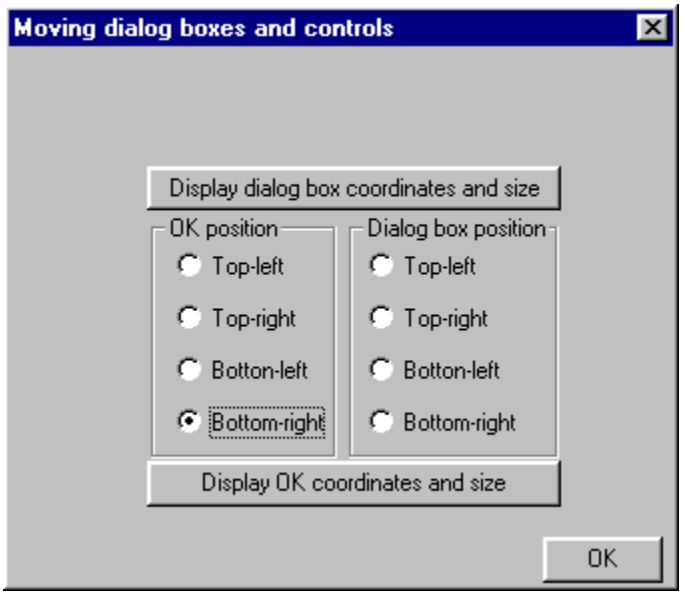
Example 5: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 221, 165, "Moving dialog boxes and controls", SUB Movement
    OKBUTTON  91, 14, 40, 14, .OK1
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  55, 62, 48, 10, .OptionButton1, "Top-left"
        OPTIONBUTTON  55, 78, 48, 10, .OptionButton2, "Top-right"
        OPTIONBUTTON  55, 94, 48, 10, .OptionButton3, "Botton-left"
        OPTIONBUTTON  55, 110, 50, 10, .OptionButton4, "Bottom-right"
    OPTIONGROUP .OptionGroup2
        OPTIONBUTTON  119, 62, 58, 10, .OptionButton5, "Top-left"
        OPTIONBUTTON  119, 78, 58, 10, .OptionButton6, "Top-right"
        OPTIONBUTTON  119, 94, 58, 10, .OptionButton7, "Botton-left"
        OPTIONBUTTON  119, 110, 58, 10, .OptionButton8, "Bottom-right"
    GROUPBOX  47, 51, 62, 75, .GroupBox2, "OK position"
    GROUPBOX  113, 51, 70, 75, .GroupBox1, "Dialog box position"
    PUSHBUTTON  46, 127, 138, 14, .PushButton1, "Display OK coordinates and size"
    PUSHBUTTON  46, 36, 138, 14, .PushButton2, "Display dialog box coordinates and size"
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB Movement(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set option button values
IF Event=0 THEN
    Dialog1.OptionButton1.SETVALUE 0
    Dialog1.OptionButton5.SETVALUE 0
ENDIF

'Moves the OK button based on an option button
IF Event=2 THEN
    IF ControlID = 3 THEN Dialog1.OK1.MOVE 4, 4, 40, 14
    IF ControlID = 4 THEN Dialog1.OK1.MOVE 178, 4, 40, 14
    IF ControlID = 5 THEN Dialog1.OK1.MOVE 4, 150, 40, 14
    IF ControlID = 6 THEN Dialog1.OK1.MOVE 178, 150, 40, 14
ENDIF

'Moves the OK button based on an option button
IF Event=2 THEN
    IF ControlID = 8 THEN Dialog1.MOVE 0, 0, 221, 165
    IF ControlID = 9 THEN Dialog1.MOVE 455, 0, 221, 165
    IF ControlID = 10 THEN Dialog1.MOVE 0, 275, 221, 165
    IF ControlID = 11 THEN Dialog1.MOVE 455, 275, 221, 165
ENDIF

'Display OK coordinates and size
IF Event=2 AND ControlID = 14 THEN
    MESSAGE "Left: " & Dialog1.OK1.GETLEFTPOSITION() & CHR(13) & "Top: "  &
Dialog1.OK1.GETTOPPOSITION() & CHR(13) & "Width: "  & Dialog1.OK1.GETWIDTH() & CHR(13) &
"Height: "  & Dialog1.OK1.GETHEIGHT()
```

```
ENDIF

'Display dialog box coordinates and size
IF Event=2 AND ControlID = 15 THEN
    MESSAGE "Left: " & Dialog1.GETLEFTPOSITION() & CHR(13) & "Top: "  &
Dialog1.GETTOPPOSITION() & CHR(13) & "Width: "  & Dialog1.GETWIDTH() & CHR(13) & "Height: "  &
Dialog1.GETHEIGHT()
ENDIF

END SUB
```

The above script displays the following dialog box. By selecting an option button, you can either move the OK button's position in the dialog box or the dialog screen placement.





Clicking the other two push buttons open a message box displaying the dialog box or the OK button coordinates.

Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

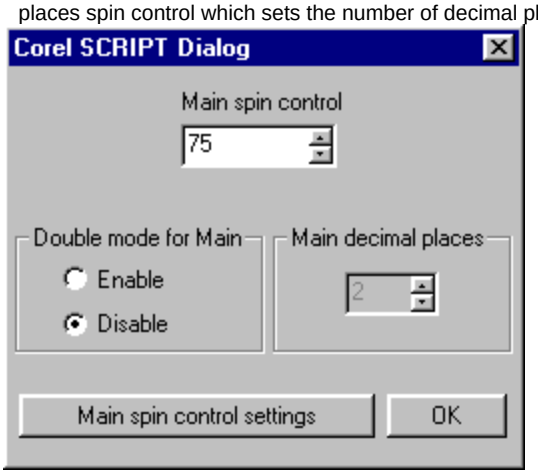
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

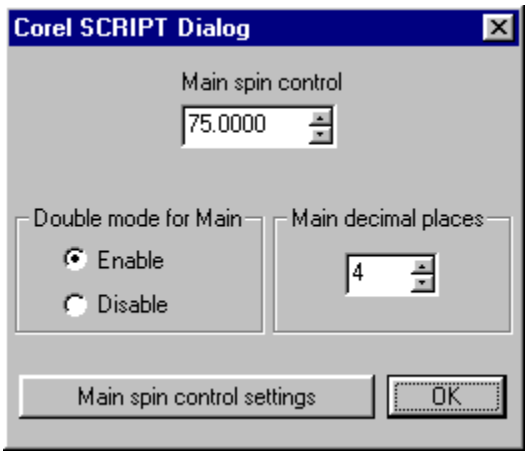
Example 6: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 170, 122, "Corel SCRIPT Dialog", SUB SPINS
    TEXT  57, 6, 56, 8, .Text2, "Main spin control"
    SPINCONTROL  57, 17, 52, 14, .SpinControl1
    GROUPBOX  88, 48, 80, 40, .GroupBox2, "Main decimal places"
    SPINCONTROL  112, 63, 31, 13, .SpinControl2
    GROUPBOX  2, 48, 83, 40, .GroupBox1, "Double mode for Main"
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  17, 60, 58, 10, .OptionButton1, "Enable"
        OPTIONBUTTON  17, 74, 58, 10, .OptionButton2, "Disable"
    OKBUTTON  126, 100, 40, 14, .OK1
    PUSHBUTTON  3, 100, 119, 14, .PushButton1, "Main spin control settings"
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB SPINS(BYVAL ControlID%, BYVAL Event%)
'Set initial Main spin control options
IF Event=0 THEN
    Dialog1.OptionButton2.SETVALUE 1
    Dialog1.SpinControl1.SETDOUBLEMODE FALSE
    Dialog1.SpinControl1.SETVALUE 75
    Dialog1.SpinControl1.SETPRECISION 0
    Dialog1.SpinControl1.SETMINRANGE 5
    Dialog1.SpinControl1.SETMAXRANGE 100
    Dialog1.SpinControl1.SETINCREMENT 5
ENDIF

'Set initial decimal spin control options
IF Event=0 THEN
    Dialog1.SpinControl2.SETVALUE 2
    Dialog1.SpinControl2.SETDOUBLEMODE FALSE
    Dialog1.SpinControl2.SETMINRANGE 0
    Dialog1.SpinControl2.SETMAXRANGE 5
    Dialog1.SpinControl2.ENABLE FALSE
ENDIF

'Enabling double mode
IF Event=2 AND ControlID=7 THEN
    IF Dialog1.OptionButton1.GETVALUE() = 1 THEN
        Dialog1.SpinControl2.ENABLE TRUE
        Dialog1.SpinControl1.SETDOUBLEMODE TRUE
        Dialog1.SpinControl1.SETPRECISION (Dialog1.SpinControl2.GETVALUE())
        Dialog1.SpinControl1.SETINCREMENT 5.55
    ENDIF
ENDIF

'Disabling double mode
```

```
IF Event=2 AND ControlID=8 THEN
    IF Dialog1.OptionButton2.GETVALUE() = 1 THEN
        Dialog1.SpinControl2.ENABLE FALSE
        Dialog1.SpinControl1.SETDOUBLEMODE FALSE
        Dialog1.SpinControl1.SETPRECISION 0
        Dialog1.SpinControl1.SETINCREMENT 5
    ENDIF
ENDIF

'Changing number of decimals
IF Event=1 AND ControlID=4 THEN
    Dialog1.SpinControl1.SETPRECISION (Dialog1.SpinControl2.GETVALUE())
ENDIF

'Main spin control settings are displayed
IF Event=2 AND ControlID=10 THEN
    MESSAGE "Minimum range: " & Dialog1.SpinControl1.GETMINRANGE()
    MESSAGE "Maximum range: " & Dialog1.SpinControl1.GETMAXRANGE()
    MESSAGE "Increment: " &  Dialog1.SpinControl1.GETINCREMENT()
    MESSAGE "Decimal places: " &  Dialog1.SpinControl1.GETPRECISION()
ENDIF

END SUB
```

The above script displays the following dialog box. By enabling double mode for the Main spin control, you enable the decimal places spin control which sets the number of decimal places in the Main spin control.





Clicking the push button opens a message box displaying a summary of the Main spin control's settings.

Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

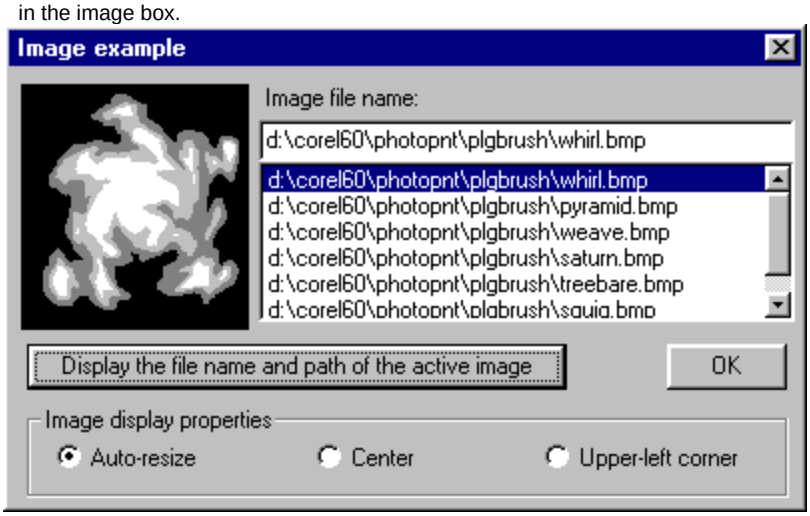
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} **Related Topics**

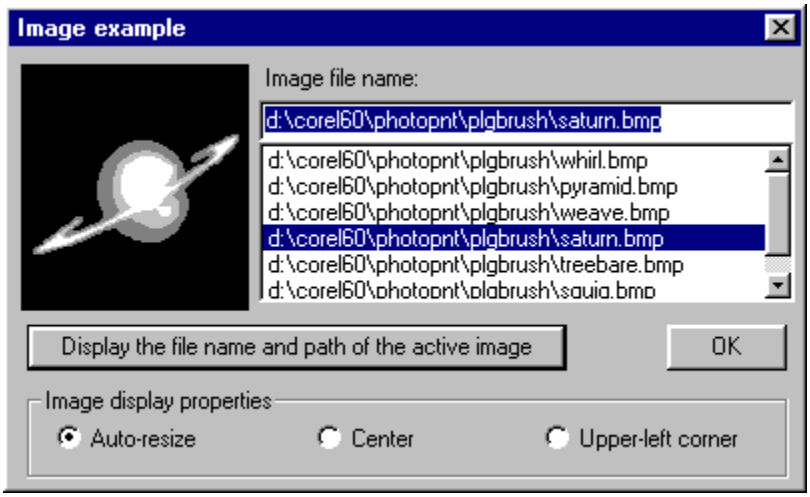
Example 7: Dynamic dialog box

```
'Set arrays for the list boxes
GLOBAL bmps(6) as string
bmps(1) = "d:\corel60\photopnt\plgbrush\whirl.bmp"
bmps(2) = "d:\corel60\photopnt\plgbrush\pyramid.bmp"
bmps(3) = "d:\corel60\photopnt\plgbrush\weave.bmp"
bmps(4) = "d:\corel60\photopnt\plgbrush\saturn.bmp"
bmps(5) = "d:\corel60\photopnt\plgbrush\treebare.bmp"
bmps(6) = "d:\corel60\photopnt\plgbrush\squig.bmp"

'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 264, 135, "Image example", SUB IMAGESUB
    PUSHBUTTON  6, 85, 180, 14, .PushButton1, "Display the file name and path of the active
image"
    IMAGE  4, 5, 75, 75, .Image1
    COMBOBOX  83, 16, 179, 64, .ComboBox1
    TEXT  85, 5, 53, 8, .Text1, "Image file name:"
    OKBUTTON  219, 85, 40, 14, .OK1
    GROUPBOX  6, 104, 254, 28, .GroupBox1, "Image display properties"
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  15, 115, 58, 10, .OptionButton1, "Auto-resize"
        OPTIONBUTTON  102, 115, 58, 10, .OptionButton2, "Center"
        OPTIONBUTTON  178, 115, 65, 10, .OptionButton3, "Upper-left corner"
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB IMAGESUB(BYVAL ControlID%, BYVAL Event%)
'initializes the dialog box
IF Event=0 THEN
    Dialog1.ComboBox1.SETARRAY bmps$
    Dialog1.Image1.SETIMAGE bmps$(1)
    'specifies the initial selection
    Dialog1.ComboBox1.SETSELECT bmps$(1)
ENDIF

'if a file is selected in the combo box (event=2) or a file is
'typed in the text box portion (event=1), the image is updated
IF Event=2 OR Event=1 AND ControlID=4 THEN
    Dialog1.Image1.SETIMAGE(Dialog1.ComboBox1.GETSELECT())
ENDIF

'Clicking push button displays image file name
IF Event=2 AND ControlID=1 THEN
    MESSAGE Dialog1.Image1.GETIMAGE()
ENDIF

'changes the image display setting
IF Event = 2 THEN
    IF Dialog1.OptionGroup1.GETVALUE() = 0 THEN
```

```
        Dialog1.Image1.SETSTYLE 16  'Auto-resize
    ELSEIF Dialog1.OptionGroup1.GETVALUE() = 1 THEN
        Dialog1.Image1.SETSTYLE 32 'Center
    ELSEIF Dialog1.OptionGroup1.GETVALUE() = 2 THEN
        Dialog1.Image1.SETSTYLE 64 'Upper-left corner
    ENDIF
ENDIF
END SUB
```

The above script displays the following dialog box. By selecting a bitmap in the combo list box you change the bitmap displayed in the image box.





Clicking the push button opens a message box displaying the name of the image in the image box. Clicking the option buttons changes the image display properties.

Note

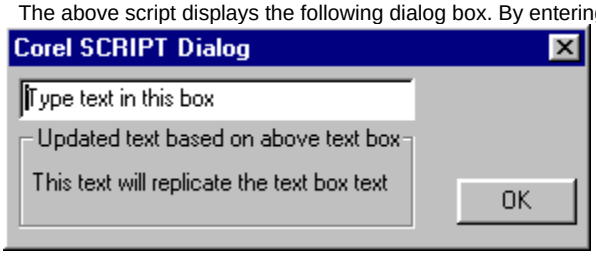
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

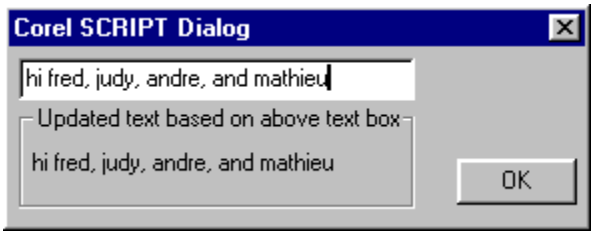
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 8: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 192, 54, "Corel SCRIPT Dialog", SUB TextEx
    TEXTBOX  3, 3, 132, 13, .TextBox1
    TEXT  7, 31, 125, 10, .Text2, "Text2"
    GROUPBOX  3, 17, 132, 32, .GroupBox1, "Updated text based on above text box"
    OKBUTTON  149, 34, 40, 14, .OK1
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB TextEx(BYVAL ControlID%, BYVAL Event%)
'initializes the dialog box
IF Event=0 THEN
    Dialog1.TextBox1.SETTEXT "Type text in this box"
    Dialog1.Text2.SETTEXT "This text will replicate the text box text"
ENDIF

'controls what happens when text is typed
IF Event=1 THEN
    Dialog1.Text2.SETTEXT(Dialog1.TextBox1.GETTEXT())
ENDIF

END SUB
```

The above script displays the following dialog box. By entering text in the text box you update the text displayed in the group box.





Note

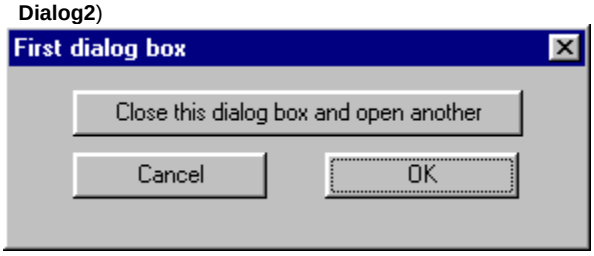
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 9: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 192, 54, "First dialog box", SUB Test1
    OKBUTTON  105, 26, 65, 14, .OK1
    CANCELBUTTON  21, 26, 65, 14, .Cancel1
    PUSHBUTTON  21, 7, 150, 14, .PushButton1, "Close this dialog box and open another"
END DIALOG

'Specifies another dialog box
BEGIN DIALOG OBJECT Dialog2 144, 44, "Second dialog box"
    OKBUTTON  40, 15, 65, 14, .OK2
END DIALOG

ReturnValue = DIALOG (Dialog1)

IF ReturnValue = 3 THEN
    DIALOG Dialog2 'opens another dialog box
ENDIF

'Dialog Event Handler subroutine
SUB Test1(BYVAL ControlID%, BYVAL Event%)

'controls what happens when push button is clicked
IF Event=2 AND ControlID=3 THEN
    Dialog1.CLOSEDIALOG 3 'closes the first dialog box
ENDIF

END SUB
```

The above script displays the following dialog box. Push buttons are often used to open a dialog box from an open dialog box. In this case, clicking the push button closes the dialog box (**CLOSEDIALOG** function) and opens another dialog box (**DIALOG Dialog2**)



Note

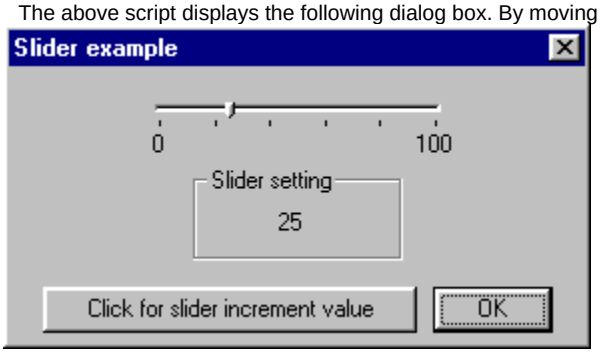
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

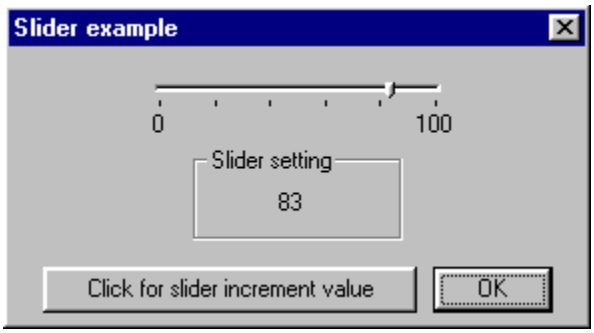
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 10: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 192, 84, "Slider example", SUB SliderSub
    HSLIDER 45, 9, 102, 9, .HSlider1
    TEXT  47, 19, 18, 8, .Text1, "0"
    TEXT  135, 19, 25, 8, .Text2, "100"
    GROUPBOX  61, 30, 70, 29, .GroupBox1, "Slider setting"
    TEXT  89, 43, 15, 8, .Text3, "25"
    PUSHBUTTON  11, 67, 125, 14, .PushButton1, "Click for slider increment value"
    OKBUTTON  141, 67, 40, 14, .OK1
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB SliderSub(BYVAL ControlID%, BYVAL Event%)

'Initialization condition to set slider values
IF Event=0 THEN
Dialog1.HSlider1.SETMINRANGE 0
    Dialog1.HSlider1.SETMAXRANGE 100
    Dialog1.HSlider1.SETINCREMENT 5
    Dialog1.HSlider1.SETVALUE 25
    Dialog1.HSlider1.SETTICK 20
ENDIF

'Following condition used when slider value changes
'Event 1 for changing slider using keyboard
'Event 2 for changing slider using mouse
IF Event=1 OR Event=2 AND ControlID=1 THEN
    Dialog1.Text3.SETTEXT (Dialog1.HSlider1.GETVALUE ())
ENDIF

'Clicking push button for increment value
IF ControlID=6 AND Event=2 THEN
    MESSAGE Dialog1.HSlider1.GETTICK ( )
ENDIF

END SUB
```

The above script displays the following dialog box. By moving the slider you change the value displayed in the group box.





Note

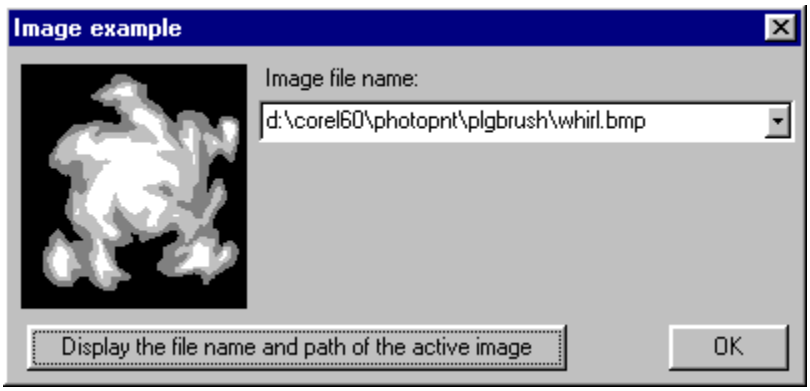
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

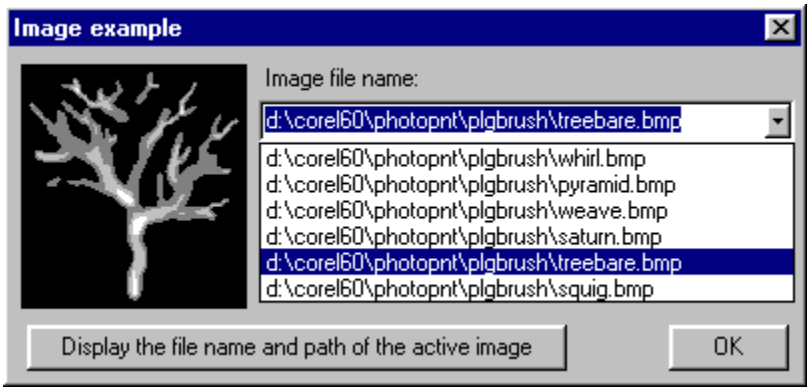
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 11: Dynamic dialog box

```
'Set arrays for the list boxes
GLOBAL bmps(6) as string
bmps(1) = "d:\corel60\photopnt\plgbrush\whirl.bmp"
bmps(2) = "d:\corel60\photopnt\plgbrush\pyramid.bmp"
bmps(3) = "d:\corel60\photopnt\plgbrush\weave.bmp"
bmps(4) = "d:\corel60\photopnt\plgbrush\saturn.bmp"
bmps(5) = "d:\corel60\photopnt\plgbrush\treebare.bmp"
bmps(6) = "d:\corel60\photopnt\plgbrush\squig.bmp"

'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 264, 102, "Image example", SUB IMAGESUB
    PUSHBUTTON  6, 85, 180, 14, .PushButton1, "Display the file name and path of the active
image"
    IMAGE  4, 5, 75, 75, .Image1
    DDCOMBOBOX  83, 16, 179, 64, .DDComboBox1
    TEXT  85, 5, 53, 8, .Text1, "Image file name:"
    OKBUTTON  220, 85, 40, 14, .OK1
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB IMAGESUB(BYVAL ControlID%, BYVAL Event%)
'initializes the dialog box
IF Event=0 THEN
    Dialog1.DDComboBox1.SETARRAY bmps$
    Dialog1.Image1.SETIMAGE bmps$(1)
    'specifies the initial selection
    Dialog1.DDComboBox1.SETSELECT bmps$(1)
ENDIF

'if a file is selected in the combo box (event=2) or a file is
'typed in the text box portion (event=1), the image is updated
IF Event=2 OR Event=1 AND ControlID=4 THEN
    Dialog1.Image1.SETIMAGE(Dialog1.DDComboBox1.GETSELECT())
ENDIF

'Clicking push button displays image file name
IF Event=2 AND ControlID=1 THEN
    MESSAGE Dialog1.Image1.GETIMAGE()
ENDIF

END SUB
```

The above script displays the following dialog box. By selecting a bitmap in the drop-down list box you change the bitmap displayed in the image box.





Clicking the push button opens a message box displaying the name of the image in the image box.

Note

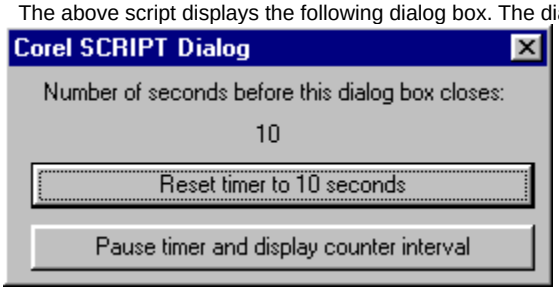
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 12: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 179, 65, "Corel SCRIPT Dialog", SUB TimerEx
    TEXT  11, 3, 156, 11, .Text1, "Number of seconds before this dialog box closes:"
    TEXT  82, 16, 10, 10, .Text2, "10" ' the number of seconds to count down
    PUSHBUTTON  7, 29, 168, 14, .PushButton1, "Reset timer to 10 seconds"
    PUSHBUTTON  7, 48, 168, 14, .PushButton2, "Pause timer and display counter interval"
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB TimerEx(BYVAL ControlID%, BYVAL Event%)
'Initialize timer interval in milliseconds
IF Event=0 THEN
    Dialog1.SETTIMER(1000) 'sets time interval to 1 second
ENDIF

'Timer event
IF Event=5 THEN
    'Passes the number of seconds to CurrentTime
    CurrentTime = Dialog1.Text2.GETTEXT()
    IF CurrentTime = 1 THEN Dialog1.CLOSEDIALOG 1
    Dialog1.Text2.SETTEXT(CurrentTime-1) 'subtracts 1 from the counter
ENDIF

'Resets timer to 10 seconds
IF Event=2 AND ControlID= 3 THEN
    Dialog1.Text2.SETTEXT "10"
ENDIF

'Pause timer and display counter interval
IF Event=2 AND ControlID= 4 THEN
    MESSAGE Dialog1.GETTIMER()/1000
ENDIF

END SUB
```

The above script displays the following dialog box. The dialog box counts down from 10 seconds and closes when it reaches 0.



Clicking the first push button resets the timer to 10 seconds. The second push button pauses the timer and opens a message box displaying timer interval.

Note

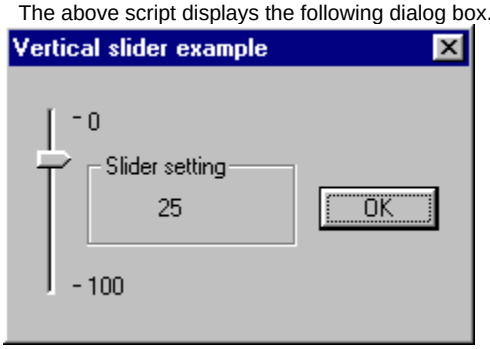
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

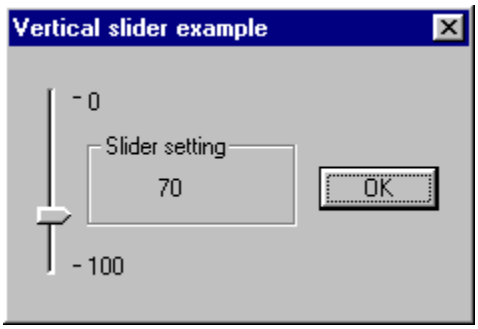
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 13: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 153, 83, "Vertical slider example", SUB SliderSub
    VSLIDER 8, 8, 17, 66, .VSlider1
    TEXT  26, 12, 18, 8, .Text1, "0"
    TEXT  26, 63, 25, 8, .Text2, "100"
    GROUPBOX  26, 26, 70, 29, .GroupBox1, "Slider setting"
    TEXT  49, 39, 15, 8, .Text3, "25"
    OKBUTTON  103, 36, 40, 14, .OK1
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB SliderSub(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set slider values
IF Event=0 THEN
    Dialog1.VSlider1.SETMINRANGE 0
    Dialog1.VSlider1.SETMAXRANGE 100
    Dialog1.VSlider1.SETINCREMENT 5
    Dialog1.VSlider1.SETVALUE 25
ENDIF

'Following condition used when slider value changes
'Event 1 for changing slider using keyboard
'Event 2 for changing slider using mouse
IF Event=1 OR Event=2 AND ControlID=1 THEN
    Dialog1.Text3.SETTEXT (Dialog1.VSlider1.GETVALUE ())
ENDIF

END SUB
```

The above script displays the following dialog box. By moving the slider you change the value displayed in the group box.





Note

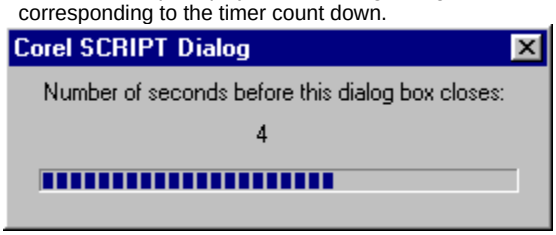
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 14: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 179, 48, "Corel SCRIPT Dialog", SUB TimerEx
    TEXT  11, 3, 156, 11, .Text1, "Number of seconds before this dialog box closes:"
    TEXT  82, 16, 10, 10, .Text2, "10"
    PROGRESS 10, 31, 160, 7, .Progress1
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB TimerEx(BYVAL ControlID%, BYVAL Event%)

'Initialize timer interval in milliseconds
IF Event=0 THEN
    Dialog1.SETTIMER(1000)
ENDIF

'Initialize timer interval in milliseconds
IF Event=0 THEN
    Dialog1.Progress1.SETMINRANGE 0
    Dialog1.Progress1.SETMAXRANGE 200
    Dialog1.Progress1.SETINCREMENT 20
ENDIF

'Timer event
IF Event=5 THEN
    'Passes the number of seconds to CurrentTime
    CurrentTime = Dialog1.Text2.GETTEXT()
    IF CurrentTime = 1 THEN Dialog1.CLOSEDIALOG 1
    Dialog1.Text2.SETTEXT(CurrentTime-1) 'subtracts 1 from the counter
    Dialog1.Progress1.STEP
ENDIF

END SUB
```

The above script displays the following dialog box. The dialog box counts down from 10 seconds and the progress bar fills corresponding to the timer count down.



Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

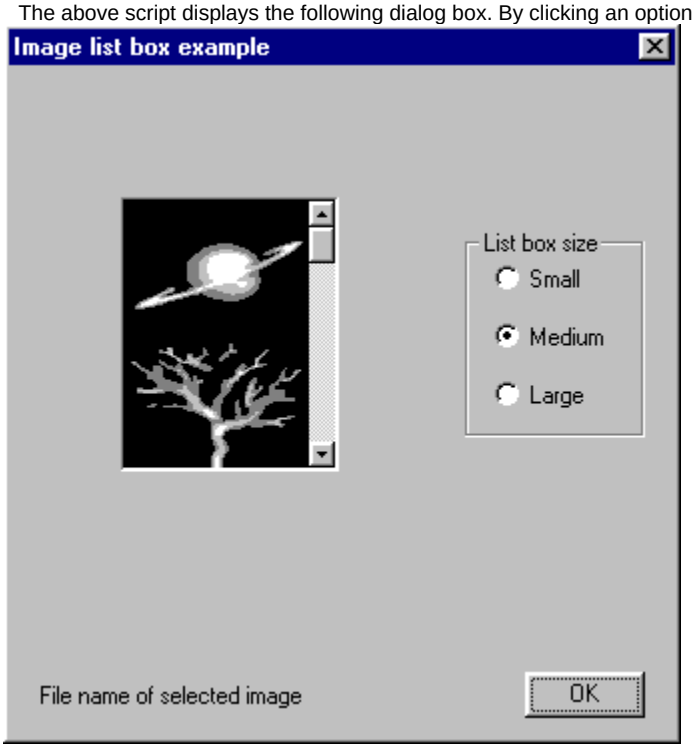
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

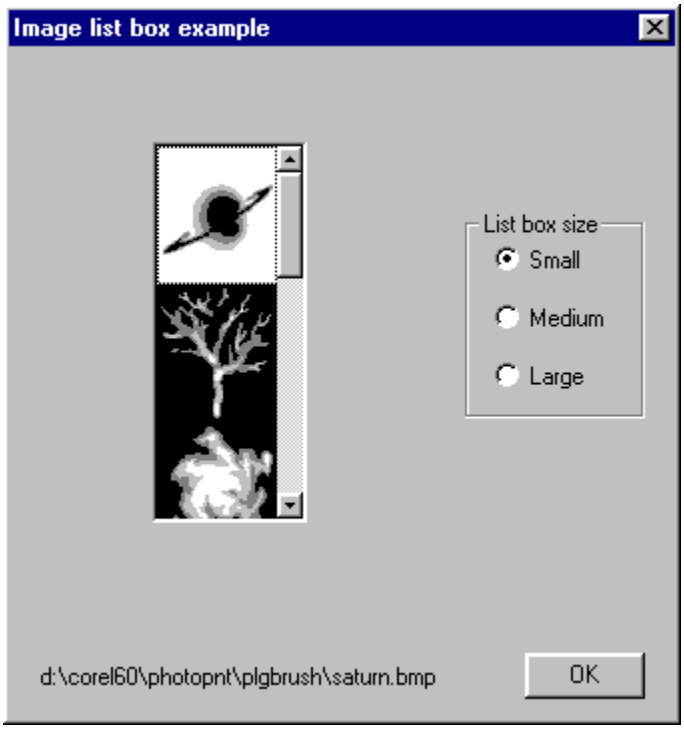
Example 15: Dynamic dialog box

```
'Set array for the image list box
GLOBAL bmps(6) as string
bmps(1) = "d:\corel60\photopnt\plgbrush\saturn.bmp"
bmps(2) = "d:\corel60\photopnt\plgbrush\treebare.bmp"
bmps(3) = "d:\corel60\photopnt\plgbrush\whirl.bmp"
bmps(4) = "d:\corel60\photopnt\plgbrush\pyramid.bmp"
bmps(5) = "d:\corel60\photopnt\plgbrush\weave.bmp"
bmps(6) = "d:\corel60\photopnt\plgbrush\squig.bmp"

'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 222, 206, "Image list box example", SUB Movement
    OKBUTTON  172, 186, 40, 14, .OK1
    GROUPBOX  152, 50, 60, 64, .GroupBox2, "List box size"
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  161, 60, 48, 10, .OptionButton1, "Small"
        OPTIONBUTTON  161, 78, 48, 10, .OptionButton2, "Medium"
        OPTIONBUTTON  161, 96, 48, 10, .OptionButton3, "Large"
    IMAGELISTBOX  37, 40, 73, 84, .ImageListBox1
    TEXT  10, 189, 155, 11, .Text1, "File name of selected image"
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB Movement(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set option button values
IF Event=0 THEN
    Dialog1.OptionButton2.SETVALUE 1
    Dialog1.ImageListBox1.SETARRAY bmps$
ENDIF

'Updates text when a image is selected
IF Event=2 AND ControlID=7 THEN
    CurrentImage$=bmps$(Dialog1.ImageListBox1.GETSELECT())
    Dialog1.Text1.SETTEXT CurrentImage$
ENDIF

'Changes the image list box to small size
IF Event=2 AND ControlID=4 THEN
    Dialog1.ImageListBox1.MOVE 48, 29, 51, 117
ENDIF

'Changes the image list box to medium size
IF Event=2 AND ControlID=5 THEN
    Dialog1.ImageListBox1.MOVE 37, 40, 73, 84
ENDIF

'Changes the image list box to large size
IF Event=2 AND ControlID=6 THEN
```

```
    Dialog1.ImageListBox1.MOVE 12, 10, 121, 176
ENDIF

END SUB
```

The above script displays the following dialog box. By clicking an option button you can change the size of the image list box.





When a image is selected, the image's file name and path is updated in the text in the bottom-left-corner of the dialog box.

Note

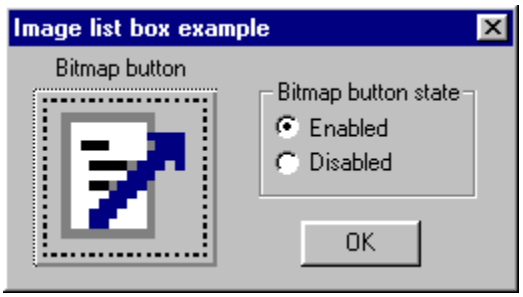
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

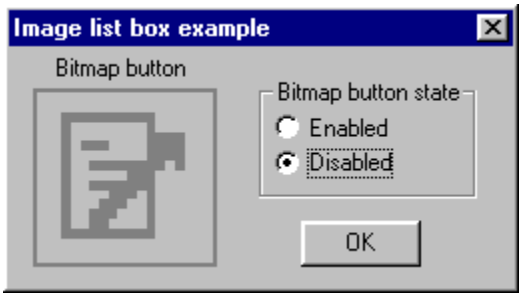
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} **Related Topics**

Example 16: Dynamic dialog box

```
'Set array for the bitmap button
GLOBAL bmps(4) as string
bmps(1) = "C:\WINDOWS\NORMAL.bmp"
bmps(2) = "C:\WINDOWS\DEPRESSED.bmp"
bmps(3) = "C:\WINDOWS\FOCUS.bmp"
bmps(4) = "C:\WINDOWS\DISABLED.bmp"

'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 168, 73, "Image list box example", SUB Movement
    BITMAPBUTTON  8, 12, 61, 55, .BitmapButton1
    OKBUTTON  97, 53, 40, 14, .OK1
    GROUPBOX  83, 9, 73, 37, .GroupBox2, "Bitmap button state"
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  88, 20, 48, 9, .OptionButton1, "Normal"
        OPTIONBUTTON  88, 31, 48, 9, .OptionButton2, "Disabled"
    TEXT  15, 2, 55, 8, .Text1, "Bitmap button"
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB Movement(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set option button values
IF Event=0 THEN
    Dialog1.OptionButton1.SETVALUE 1
    Dialog1.BitmapButton1.SETARRAY bmps$
ENDIF

'Displays a text box when bitmap button is clicked
IF Event=2 AND ControlID=1 THEN
    MESSAGE "You pressed the bitmap button!"
ENDIF

'Enables the bitmap button
IF Event=2 AND ControlID=5 THEN
    Dialog1.BitmapButton1.ENABLE TRUE
ENDIF

'Disables the bitmap button
IF Event=2 AND ControlID=6 THEN
    Dialog1.BitmapButton1.ENABLE FALSE
ENDIF

END SUB
```

The above script displays the following dialog box. By clicking an option button you can enable and disable the bitmap button.





Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

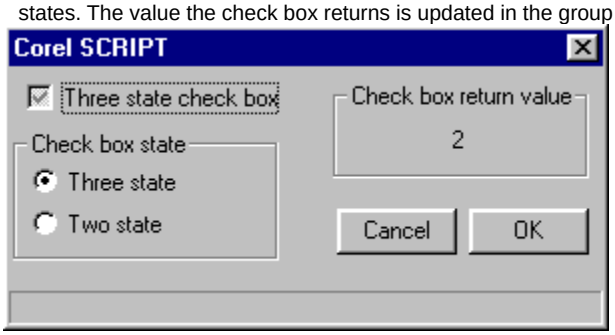
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

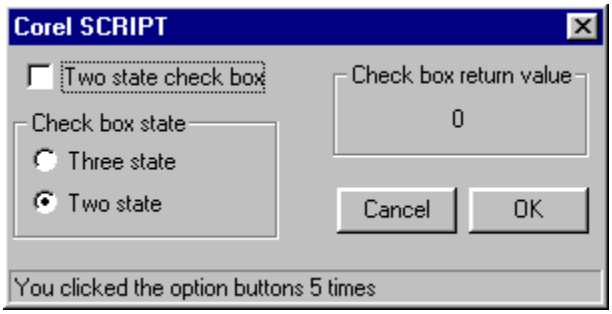
Example 17: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 198, 79, "Corel SCRIPT", SUB CS_EX
    CHECKBOX  6, 4, 100, 10, .CheckBox1, "Three state check box"
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  7, 30, 60, 10, .OptionButton1, "Three state"
        OPTIONBUTTON  7, 43, 48, 10, .OptionButton2, "Two state"
    GROUPBOX  1, 19, 89, 40, .GroupBox1, "Check box state"
    OKBUTTON  153, 43, 40, 14, .OK1
    CANCELBUTTON  109, 43, 40, 14, .Cancel1
    GROUPBOX  108, 4, 85, 30, .GroupBox2, "Check box return value"
    TEXT  147, 18, 10, 8, .Text1, "2"
    STATUS .Status1
END DIALOG
Dialog_Return_Value% = DIALOG (Dialog1)

'Pressing the OK button
IF Dialog_Return_Value%=1 THEN
    MESSAGE "Closes this dialog box and saves any changes you have made."
ENDIF

'Pressing the Cancel button
IF Dialog_Return_Value%=2 THEN
    MESSAGE "Closes this dialog box without saving any changes you have made."
ENDIF

GLOBAL Status_Counter%

'Dialog Event Handler subroutine
SUB CS_EX(BYVAL ControlID%, BYVAL Event%)
'Initialization condition for check box and option button
IF Event=0 THEN
    Dialog1.OptionButton1.SETVALUE 1
    Dialog1.CheckBox1.SETVALUE 2
ENDIF

'Clicking the two state option button
IF Event=2 AND ControlID=4 THEN
    Dialog1.CheckBox1.SETTHREESTATE FALSE
    Dialog1.CheckBox1.SETTEXT "Two state check box"
    'next two line set the text value to the checkbox value
    CheckVal$ = Dialog1.CheckBox1.GETVALUE()
    Dialog1.Text1.SETTEXT CheckVal$
ENDIF

'Clicking the two state option button
IF Event=2 AND ControlID=5 THEN
    Dialog1.CheckBox1.SETTHREESTATE TRUE
    Dialog1.CheckBox1.SETTEXT "Two state check box"
```

```
    'next two line set the text value to the checkbox value
    CheckVal$ = Dialog1.CheckBox1.GETVALUE()
    Dialog1.Text1.SETTEXT CheckVal$
ENDIF

'Clicking the checkbox changes the text
IF Event=2 AND ControlID=1 THEN
    CheckVal$ = Dialog1.CheckBox1.GETVALUE()
    Dialog1.Text1.SETTEXT CheckVal$
ENDIF

'Clicking an option button changes status bar text
IF Event=2 AND (ControlID=3 OR ControlID=4) THEN
    Status_Counter% = 1 + Status_Counter% ' this is a counter
    Dialog1.Status1.SETTEXT "You clicked the option buttons " & Status_Counter% & " times"
ENDIF

END SUB
```

The above script displays the following dialog box. By clicking an option button you can make the check box have two or three states. The value the check box returns is updated in the group box as you click it.





Each time you click an option button the text in the status bar changes. When the dialog box is closed, a message box is displayed letting you know which button was used to close it.

Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

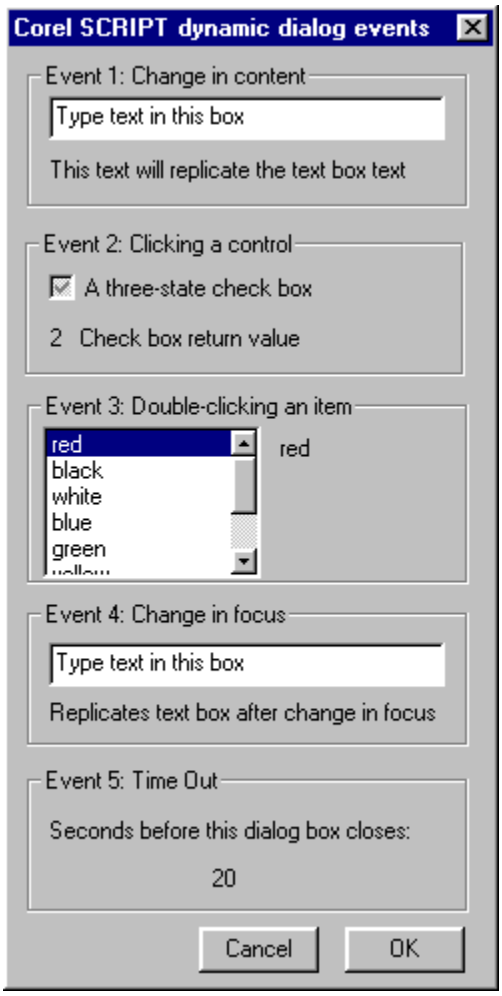
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 18: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 161, 288, "Corel SCRIPT dynamic dialog events", SUB CS_EX
    GROUPBOX  6, 4, 147, 45, .GroupBox4, "Event 1: Change in content"
    TEXTBOX  13, 15, 132, 13, .TextBox1
    TEXT  13, 33, 132, 10, .Text4, "Text4"

    GROUPBOX  5, 56, 147, 45, .GroupBox2, "Event 2: Clicking a control"
    CHECKBOX  13, 69, 100, 10, .CheckBox1, "A three-state check box"
    TEXT  13, 85, 7, 8, .Text3, "2"
    TEXT  23, 85, 100, 8, .Text5, "Check box return value"

    GROUPBOX  6, 106, 147, 59, .GroupBox5, "Event 3: Double-clicking an item"
    LISTBOX  11, 116, 73, 47, .ListBox1
    TEXT  90, 119, 50, 8, .Text7, "red"

    GROUPBOX  6, 170, 147, 45, .GroupBox1, "Event 4: Change in focus"
    TEXTBOX  13, 183, 132, 13, .TextBox2
    TEXT  13, 200, 132, 8, .Text6, "Text6"

    GROUPBOX  6, 221, 147, 45, .GroupBox3, "Event 5: Time Out"
    TEXT  13, 236, 122, 11, .Text1, "Seconds before this dialog box closes:"
    TEXT  67, 251, 10, 10, .Text2, "20"      ' the number of seconds to count down

    OKBUTTON  112, 270, 40, 14, .OK1
    CANCELBUTTON  63, 270, 40, 14, .Cancel1
END DIALOG

GLOBAL color$(8)
color(1) = "red"
color(2) = "black"
color(3) = "white"
color(4) = "blue"
color(5) = "green"
color(6) = "yellow"
color(7) = "purple"
color(8) = "brown"

Dialog_Return_Value% = DIALOG (Dialog1)


'Dialog Event Handler subroutine
SUB CS_EX(BYVAL ControlID%, BYVAL Event%)

'Dialog initialization event
IF Event=0 THEN
    'Event 1 group box controls
    Dialog1.TextBox1.SETTEXT "Type text in this box"
    Dialog1.Text4.SETTEXT "This text will replicate the text box text"
```

```
    'Event 2 group box controls
    Dialog1.CheckBox1.SETVALUE 2
    'Event 3 list box control
    Dialog1.ListBox1.SETARRAY color$
    Dialog1.ListBox1.SETSELECT 1
    'Event 4 group box controls
    Dialog1.TextBox2.SETTEXT "Type text in this box"
    Dialog1.Text6.SETTEXT "Replicates text box after change in focus"
    'Event 5 group box controls
    Dialog1.SETTIMER(1000) 'sets time interval to 1 second
ENDIF

'Change in content event
IF Event=1 THEN
    Dialog1.Text4.SETTEXT(Dialog1.TextBox1.GETTEXT())
ENDIF

'Clicking a control
IF Event=2 THEN
    Dialog1.Text3.SETTEXT(Dialog1.CheckBox1.GETVALUE())
ENDIF

'Double-clicking a list box item
IF Event=3 THEN
    Dialog1.Text7.SETTEXT color$(Dialog1.ListBox1.GETSELECT())
ENDIF

'Change in focus
IF Event=4 THEN
    Dialog1.Text6.SETTEXT(Dialog1.TextBox2.GETTEXT())
ENDIF

'Time out event
IF Event=5 THEN
    'Passes the number of seconds to CurrentTime
    CurrentTime = Dialog1.Text2.GETTEXT()
    IF CurrentTime = 1 THEN Dialog1.CLOSEDIALOG 1
    Dialog1.Text2.SETTEXT(CurrentTime-1) 'subtracts 1 from the counter
ENDIF

END SUB
```

The above script provides an example of all 6 dynamic dialog box events as shown in the following dialog box.



- The Dialog Initialization event (0) sets the initial values for many of the dialog controls.
- The Change in content event (1) is shown when text is entered in the top-most text box.
- The Clicking a control event (2) is shown when check box is clicked.
- The Double-clicking a list box item event (3) is shown when an item is double-clicked.
- The Change in focus event (4) is shown after text is entered in the second text box and dialog box focus is sent to another control.
- The Time out event (5) is shown with a count down before the dialog box is closed.

Note

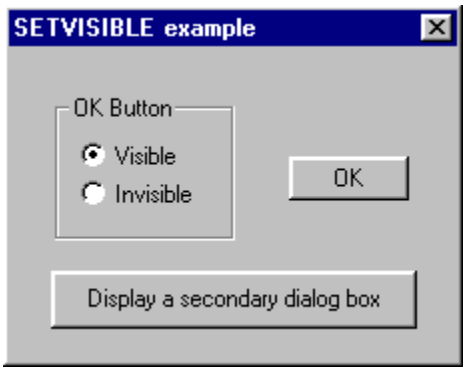
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} **Related Topics**

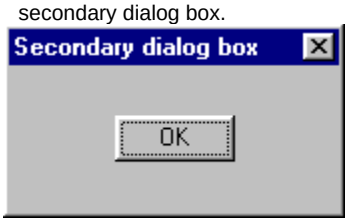
Example 19: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 149, 96, "SETVISIBLE example", SUB HideDisplay
    GROUPBOX  15, 14, 60, 45, .GroupBox1, "OK Button"
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  23, 28, 45, 10, .OptionButton1, "Visible"
        OPTIONBUTTON  23, 40, 45, 10, .OptionButton2, "Invisible"
    OKBUTTON  93, 33, 40, 14, .OK1
    PUSHBUTTON  14, 68, 122, 18, .PushButton1, "Display a secondary dialog box"
END DIALOG

'Specifies a secondary dialog box
BEGIN DIALOG Dialog2 110, 44, "Secondary dialog box"
    OKBUTTON  35, 15, 40, 14
END DIALOG

DIALOG Dialog1 'Displays the first dialog box

'Dialog Event Handler subroutine
SUB HideDisplay(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set option button value
IF Event=0 THEN
    Dialog1.OptionButton1.SETVALUE 1
ENDIF

'controls whether to display the OK button
IF Event=2 THEN
    IF ControlID=4 THEN
        Dialog1.OK1.SETVISIBLE FALSE
    ELSEIF ControlID=3 THEN
        Dialog1.OK1.SETVISIBLE TRUE
    ENDIF
ENDIF

'controls what happens when the push button is clicked
IF Event=2 AND ControlID=6 THEN
    Dialog1.SETVISIBLE FALSE 'hides main dialog box
    DIALOG Dialog2 'displays secondary dialog box
    Dialog1.SETVISIBLE TRUE 'redisplays main dialog box
ENDIF

END SUB
```

The above script displays the following dialog box.



Clicking an option button hides or displays the OK button. Clicking the push button hides the dialog box and displays a secondary dialog box.



Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

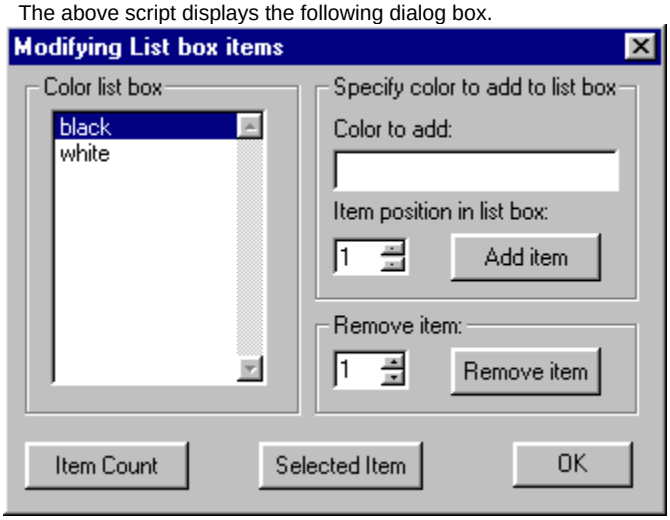
{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

Example 20: Dynamic dialog box

```
'Specifies the dynamic dialog box
'Sets the array for the list box
GLOBAL color$(2)
color(1) = "black"
color(2) = "white"

'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 217, 136, "Modifying List box items", SUB ListItems
    GROUPBOX  5, 2, 92, 105, .GroupBox1, "Color list box"
    LISTBOX  14, 13, 72, 85, .ListBox1
    GROUPBOX  102, 2, 108, 70, .GroupBox2, "Specify color to add to list box"
    TEXT  108, 15, 50, 8, .Text1, "Color to add:"
    TEXTBOX  108, 25, 95, 13, .TextBox1
    TEXT  108, 40, 85, 8, .Text2, "Item position in list box:"
    SPINCONTROL  108, 52, 25, 12, .SpinControl1
    PUSHBUTTON  147, 51, 50, 15, .PushButton1, "Add item"
    GROUPBOX  102, 75, 108, 32, .GroupBox3, "Remove item:"
    SPINCONTROL  108, 87, 25, 12, .SpinControl2
    PUSHBUTTON  147, 86, 50, 15, .PushButton2, "Remove item"
    PUSHBUTTON  59, 115, 48, 15, .PushButton3, "Selected Item"
    PUSHBUTTON  5, 115, 48, 15, .PushButton4, "Item Count"
    PUSHBUTTON  113, 115, 48, 15, .PushButton5, "Empty List"
    OKBUTTON  168, 115, 40, 14, .OK1
END DIALOG
DIALOG Dialog1 'displays the dialog box

'Dialog Event Handler subroutine
SUB ListItems(BYVAL ControlID%, BYVAL Event%)
'Initialization condition to set option button values
IF Event=0 THEN
    Dialog1.ListBox1.SETARRAY Color$
    Dialog1.ListBox1.SETSELECT 1
    'Set the add spin control range
    Dialog1.SpinControl1.SETMINRANGE 1
    Dialog1.SpinControl1.SETMAXRANGE 3
    'Set the remove spin control range
    Dialog1.SpinControl2.SETMINRANGE 1
    Dialog1.SpinControl2.SETMAXRANGE 2
ENDIF

'Selected item push button (uses the GETID function)
IF ControlID = Dialog1.PushButton3.GETID ( ) AND Event = 2 THEN
    Color_Num = Dialog1.ListBox1.GETSELECT()
    MESSAGE Dialog1.ListBox1.GETITEM(Color_Num)
ENDIF

'Item Count push button (uses the GETID function)
IF ControlID = Dialog1.PushButton4.GETID ( ) AND Event = 2 THEN
```

```
    MESSAGE Dialog1.ListBox1.GETITEMCOUNT()
ENDIF

'Empty List push button (uses the GETID function)
IF ControlID = Dialog1.PushButton5.GETID ( ) AND Event = 2 THEN
    Dialog1.ListBox1.RESET
'Set the spin control ranges
    Dialog1.SpinControl1.SETMAXRANGE (Dialog1.ListBox1.GETITEMCOUNT() + 1)
    Dialog1.SpinControl2.SETMAXRANGE (Dialog1.ListBox1.GETITEMCOUNT())
    Dialog1.SpinControl1.SETMINRANGE 0
    Dialog1.SpinControl2.SETMINRANGE 0
ENDIF

'Add item push button
IF ControlID = 8 AND Event = 2 THEN
    Item_text=Dialog1.Textbox1.GETTEXT()
    Item_num=Dialog1.SpinControl1.GETVALUE()
    Dialog1.ListBox1.ADDITEM  Item_text, Item_num
    'Set the spin control ranges
    Dialog1.SpinControl1.SETMAXRANGE (Dialog1.ListBox1.GETITEMCOUNT() + 1)
    Dialog1.SpinControl2.SETMAXRANGE (Dialog1.ListBox1.GETITEMCOUNT())
ENDIF

'Remove item push button
IF ControlID = 11 AND Event = 2 THEN
    Delete_item=Dialog1.SpinControl2.GETVALUE()
    Dialog1.ListBox1.REMOVEITEM Delete_item
    'Set the spin control ranges
    Dialog1.SpinControl1.SETMAXRANGE (Dialog1.ListBox1.GETITEMCOUNT() + 1)
    Dialog1.SpinControl2.SETMAXRANGE (Dialog1.ListBox1.GETITEMCOUNT())
    Dialog1.ListBox1.SETSELECT 1    'Selects the first list item
ENDIF

END SUB
```

The above script displays the following dialog box.



This dialog box performs the following functions:

- The **Item Count** button displays a message box noting the number of items in the list box.
- The **Selected Item** button displays a message box noting the selected item in the list box.
- The **Empty List** button removes all the items from the list box.
- The **Remove Item** group box is used to remove an item from the list box. To remove an item, specify the item to remove in the spin control and click the **Remove Item** push button.
- The **Specify color to add to list box** group box is used to add an item to the list box. To add an item, specify the item in the text box, specify its position in the spin control and click the **Add Item** push button.

Note

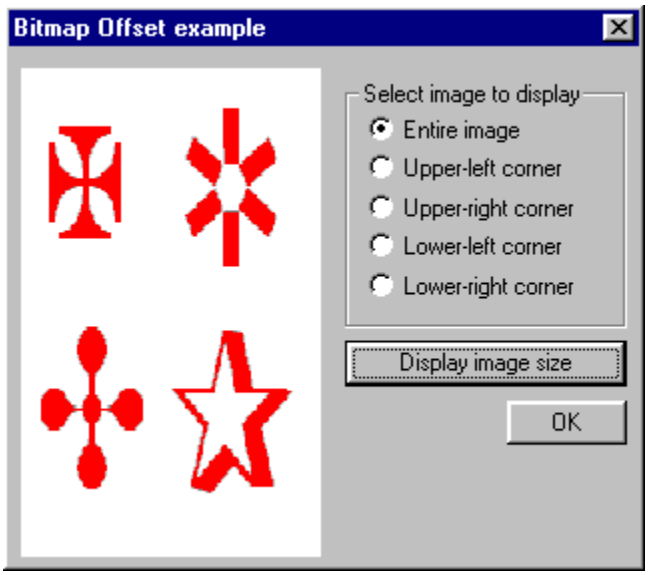
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

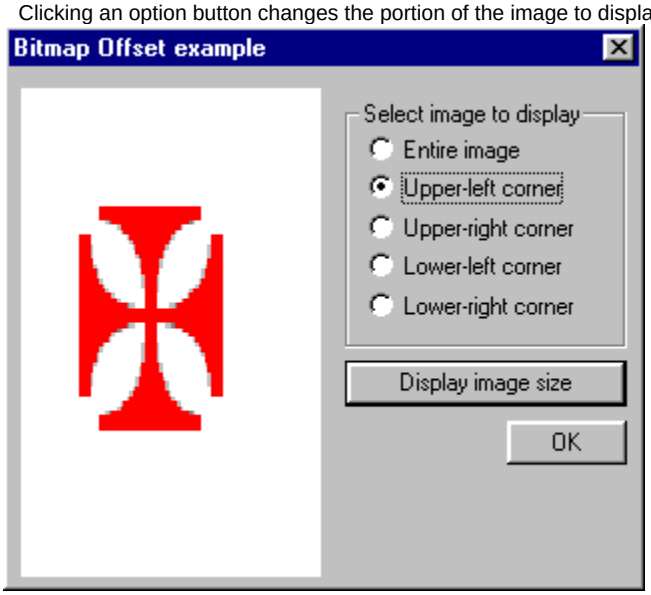
Example 21: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 210, 159, "Bitmap Offset example", SUB IMAGESUB
    PUSHBUTTON  112, 90, 94, 14, .PushButton1, "Display image size"
    IMAGE  4, 6, 100, 150, .Image1
    OKBUTTON  166, 108, 40, 14, .OK1
    GROUPBOX  112, 10, 94, 76, .GroupBox1, "Select image to display"
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON  119, 20, 58, 10, .OptionButton1, "Entire image"
        OPTIONBUTTON  119, 32, 75, 10, .OptionButton2, "Upper-left corner"
        OPTIONBUTTON  119, 44, 75, 10, .OptionButton3, "Upper-right corner"
        OPTIONBUTTON  119, 56, 75, 10, .OptionButton4, "Lower-left corner"
        OPTIONBUTTON  119, 68, 75, 10, .OptionButton5, "Lower-right corner"
END DIALOG
DIALOG Dialog1

'Dialog Event Handler subroutine
SUB IMAGESUB(BYVAL ControlID%, BYVAL Event%)
'initializes the dialog box
IF Event=0 THEN
    'sets the bitmap (200 pixels by 200 pixels)
    Dialog1.Image1.SETIMAGE "C:\windows\desktop\big_four.bmp"
ENDIF

'Clicking push button displays image size in pixels
IF Event=2 AND ControlID=1 THEN
    MESSAGE "Bitmap height: " & Dialog1.Image1.GETBITMAPHEIGHT()
    MESSAGE "Bitmap width: " & Dialog1.Image1.GETBITMAPWIDTH()
ENDIF

'changes the image display setting
IF Event = 2 THEN
    IF Dialog1.OptionGroup1.GETVALUE() = 0 THEN
        Dialog1.Image1.SETBITMAPOFFSET 0, 0
    ELSEIF Dialog1.OptionGroup1.GETVALUE() = 1 THEN
        Dialog1.Image1.SETBITMAPOFFSET 0, 0, 100, 100
    ELSEIF Dialog1.OptionGroup1.GETVALUE() = 2 THEN
        Dialog1.Image1.SETBITMAPOFFSET 100, 0, 200, 100
    ELSEIF Dialog1.OptionGroup1.GETVALUE() = 3 THEN
        Dialog1.Image1.SETBITMAPOFFSET 0, 100, 100, 200
    ELSEIF Dialog1.OptionGroup1.GETVALUE() = 4 THEN
        Dialog1.Image1.SETBITMAPOFFSET 100, 100, 200, 200
    ENDIF
ENDIF
END SUB
```

The above script displays the following dialog box.



Clicking an option button changes the portion of the image to display in the image box.



Note

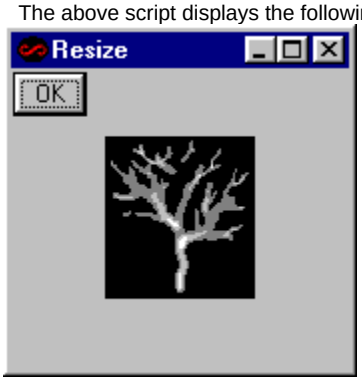
- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

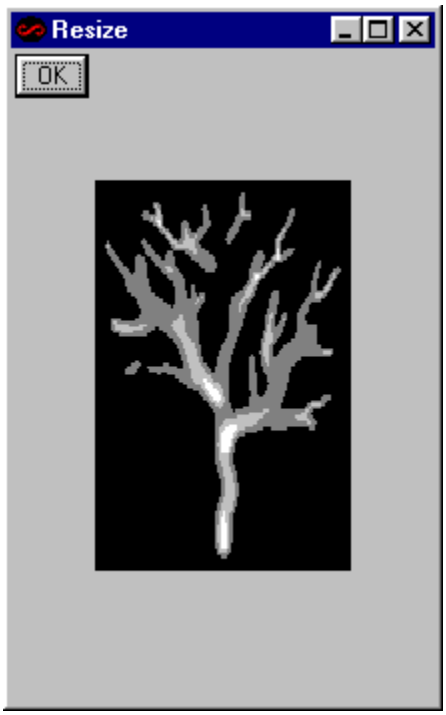
Example 22: Dynamic dialog box

```
'Specifies the dynamic dialog box
BEGIN DIALOG OBJECT Dialog1 114, 93, "Resize", SUB ResizeDB
    OKBUTTON  1, 1, 25, 14, .OK1
    IMAGE  32, 21, 50, 50, .Image1
END DIALOG

'the next line sets image
Dialog1.Image1.SETIMAGE "d:\corel60\photopnt\plgbrush\treebare.bmp"
Dialog1.SETSTYLE 64 'sets dialog box style to resize

DIALOG Dialog1

'Dialog Event Handler subroutine
SUB ResizeDB(BYVAL ControlID%, BYVAL Event%)

'changes size and position of image box when dialog is resized
IF Event = 6 THEN
    DW = Dialog1.GETWIDTH ( ) 'returns width of dialog box
    DH = Dialog1.GETHEIGHT ( ) 'returns height of dialog box
    Dialog1.Image1.MOVE 0.2*DW, 0.2*DH, 0.6*DW, 0.6*DH 'resizes image control
ENDIF

END SUB
```

The above script displays the following dialog box.



By resizing the dialog box, the image control is also resized. The width and height of the image control is set to 60% of the width and height of the dialog box.



Note

- You can paste the above script into the Corel SCRIPT Editor by selecting it with the mouse, right-clicking and selecting Copy. The script can then be pasted into the Corel SCRIPT Editor.

{button ,AL(`dialog_example_all_dyn;;;;;',0,"Defaultoverview",)} Related Topics

ADDFOL statement

#ADDFOL folder

This statement adds a temporary folder to the paths Corel SCRIPT searches when trying to find an **INCLUDE** file on your system.

Parameter	Description
folder	String expression specifying a drive and a folder.

Note

- The pound sign (#) is required in the syntax.
- Corel SCRIPT searches for an INCLUDE file in the following order:

1. The folder where the script resides. You can use the **GETSCRIPTFOLDER** statement to set or determine the active folder.
2. Folders in the path. The path is specified in the systems AUTOEXEC.BAT file.
3. Folders set in the Corel SCRIPT Editors **INCLUDE** option. Click ➪ for more information about setting INCLUDE folders.
4. Folders specified with the ADDFOL statement.

Example

```
#ADDFOL "C:\MyFiles"
```

The above example add the MYFILES folder on the C drive to the path Corel SCRIPT searches for INCLUDE files.

{button ,AL(`include;setcurrfolder;;;;',0,"Defaultoverview",)} Related Topics

ADDRESBMP statement

#ADDRESBMP name file

This statement embeds a Windows bitmap graphic (.BMP) into an executable (.EXE), DLL, or Corel Add-on (.CAO) created with Corel SCRIPT or a Corel SCRIPT Binary file (.CSB). If you're distributing a script that uses many bitmaps in the form of an executable, DLL, or Corel SCRIPT Binary file, this statement can reduce the number of files that you must distribute.

In Corel SCRIPT, bitmaps are most often used in custom dialog boxes. See the Image dialog box control for an example of a script using bitmaps.

Parameter	Description
name	String expression specifying the bitmap reference name within a script. When this name is referenced within a script, it must be preceded with the pound sign (#) and be enclosed in quotations. However, within the **ADDRESBMP** statement, quotations should not be used. See the example below for more information.
file	String expression specifying the filename and path of a Windows bitmap graphic.

Note

- The pound sign (#) is required in the syntax.

Example

```
#ADDRESBMP Wizard1Portrait "C:\MyScripts\portrait.bmp"
BEGIN DIALOG Dialog1 200, 100, "Corel SCRIPT Dialog"
    IMAGE  11, 17, 74, 65, "#Wizard1Portrait"
END DIALOG
```

In the above example, the PORTRAIT.BMP bitmap file takes on the reference name Wizard1Portrait. This bitmap is then used in an image control in a custom dialog box. If this script was compiled into an executable the PORTRAIT bitmap would be embedded into the executable file.

{button ,AL(`image;image_dyn;Creating Corel SCRIPT Executables;Distributing_cs_exe;;',0,"Defaultoverview",)}
Related Topics

BEEP statement

BEEP

Sounds a tone.

Note

- The sound your computer makes depends on your computer's hardware (for example, sound cards, PC speaker, and so on) and the Default sound in your Windows sound settings (see your Windows Control Panel for more details).

Example

```
IF (abc<=15.3) then BEEP ELSE MESSAGE "It's greater than 15.3"
```

If the variable **abc** is less than or equal to 15.3, the computer sounds a tone.

{button ,AL(`csui_statements;;;;;',0,"Defaultoverview",)} **Related Topics**

BEGINWAITCURSOR and ENDWAITCURSOR statements

BEGINWAITCURSOR
ENDWAITCURSOR

The **BEGINWAITCURSOR** statement sets the mouse pointer to Busy. The Busy pointer usually appears as an hour-glass on most systems. This command is useful for scripts that perform long operations. By setting the pointer to Busy, the user is alerted that the script is still executing.

The **ENDWAITCURSOR** statement returns the pointer to its normal state. If the **ENDWAITCURSOR** statement is not in a script that uses the **BEGINWAITCURSOR** statement, the pointer reverts to its normal state after the script finishes executing.

Note

- It is good programming practice to use an **ENDWAITCURSOR** statement with every **BEGINWAITCURSOR** statement.

Example

```
BEGINWAITCURSOR
WAIT FOR 30
ENDWAITCURSOR
```

In the above example, the pointer is set to Busy, script execution is paused for 30 seconds, and then the pointer reverts to its normal state.

{button ,AL(`csui_statements;;;;;',0,"Defaultoverview",)} Related Topics

COPY statement and function

COPY file1, file2, overwrite

The **COPY** statement copies a file.

You can also use **COPY** as a function: it returns TRUE (-1) if the **COPY** operation is successful; **FALSE** (0) otherwise.

Parameter	Description
file1	String expression specifying the file to copy. **file1** can include the drive and folder.
file2	String expression specifying the where **file1** is to be copied. **file2** can include the drive and folder.
overwrite	If **file2** already exists, this determines whether to overwrite the existing file: 0 copy and overwrite (default if omitted) 1 overwrite fails

Example

```
DIM x AS STRING
DIM y AS STRING
x = "C:\work\example1.cdr"
y = "D:\work\example1.cdr"
COPY x, y, 0
```

The above example copies the EXAMPLE1.CDR file to the work folder on the D drive.

```
DIM x AS STRING
DIM y AS STRING
x = "C:\work\example1.cdr"
y = "D:\work\example1.cdr"
success = COPY (x, y, 0)
```

The above example copies the EXAMPLE1.CDR file to the Work folder on the D drive, and assigns -1 to **success**.

{button ,AL(`FINDFIRSTFOLDER_FINDNEXTFOLDER;rename;copy;SETCURRFOLDER;rmfolder;GETcurrfolder;;',0,"Defaultoverview",)} Related Topics

FILEATTR function

retval = FILEATTR(FileName)

This functions returns a file's or folder's attributes.

Return Value

The FILEATTR function returns one of the following values:

- 0⇨file doesn't exist
- 1⇨read-only files or folder
- 2⇨hidden files or folder
- 4⇨system files or folder used by operating system
- 16⇨folder
- 32⇨archive files or folder
- 128⇨normal files or folder
- 256⇨temporary files or folder
- 2048⇨compressed files and folders

Parameter	Description
FileName	String expression specifying the file/folder for which the attributes are returned.

Example

```
retval = FILEATTR("C:\myfiles\mysetup.txt")
```

If MYSETUP.TXT is read-only, hidden, and a system file, **retval** equals 7.

In cases where multiple attributes are returned, you can use the AND (bitwise) operator to determine specific attributes. To determine if MYSETUP.TXT was a read-only file you could use the following syntax:

```
IF 1 AND retval THEN readOnly$ = "Yes" ELSE readOnly$ = "No"
```

1 is the read-only attribute. The variable **readOnly** is assigned a string based on bitwise comparison. In this case **readOnly** is assigned "Yes".

{button ,AL(`GETCURRFOLDER;filemode;getfileattr;FINDFIRSTFOLDER_FINDNEXTFOLDER;setcurrfolder;',0,"Defaultoverview",)} Related Topics

FILEDATE function

retval = FILEDATE(FileName)

This functions returns a file's last modification date.

Return Value

Assigned the date of the specified file's last modification as date data type. If the file is not found, 0 is returned.

Parameter	Description
FileName	Lets you specify the file for which the date property is returned.

Note

- If the file is not closed when the function is executed, the function returns 12:00 AM.

Example

```
retval = FILEDATE("C:\myfiles\mytext.txt")
```

{button ,AL(`GETCURRFOLDER;filemode;getfileattr;FINDFIRSTFOLDER_FINDNEXTFOLDER;setcurrfolder;',0,"Defaultoverview",)} Related Topics

FILEMODE function

FILEMODE(num)

Returns the file mode of an open text file in your computer's memory.

Parameter	Description
num	Numeric expression specifying an open file number to examine: 1 file was opened for input 2 file was opened for output 8 file was opened for append

Note

- This the only Corel SCRIPT file function that doesn't have an optional # sign in front of the file number.

Examples

```
i% = FILEMODE(1)
```

If file 1 was opened for input, then **i%** is set to 1. If file 1 was opened for output, then **i%** is set to 2. If file 1 was opened for append, then **i%** is set to 8.

```
OPEN "C:\example.txt" FOR APPEND AS 2
i% = FILEMODE(2)
```

Assigns 8 to the variable **i%**.

{button ,AL(`OPEN_APPEND;OPEN_INPUT;OPEN_OUTPUT;;;',0,"Defaultoverview",)} Related Topics

FILEPOS function

FILEPOS (#num)

Returns the current file position of the file pointer for the specified file.

Parameter	Description
#**num**	Numeric expression specifying an open file number to examine. The # sign is optional.

Note

- The seek always starts from the first position in the file.

Example

```
OPEN "C:\HELLO.TXT" FOR INPUT AS 2
SEEK 2, 12
position% = FILEPOS(2)
```

Assigns 12 to the variable **i%**.

{button ,AL(`lof;seek;open_append;print;write;',0,"Defaultoverview",)} Related Topics

FILESIZE function

retval = FILESIZE(FileName)

Use the FILESIZE function to return a file's size in bytes.

Return Value

Assigned the size of the specified file in bytes. If the file is not found, 0 is returned.

Parameter	Description
FileName	Lets you specify the file for which the date property is returned.

Example

```
retval = FILESIZE("C:\myfiles\mytext.txt")
```

{button ,AL(`GETCURRFOLDER;filemode;getfileattr;FINDFIRSTFOLDER_FINDNEXTFOLDER;setcurrfolder;',0,"Defaultoverview",)} Related Topics

FINDFIRSTFOLDER, FINDNEXTFOLDER functions

FolderFileName$ = FINDFIRSTFOLDER(searchcriteria, attributes)
FolderFileName$ = FINDNEXTFOLDER( )

Use the **FINDFIRSTFOLDER** and **FINDNEXTFOLDER** functions to assemble or perform an operation on a list of files, folders, or both. The **FINDFIRSTFOLDER** function is used to locate the first file or first folder in a folder that meets a specified search criteria. The **FINDNEXTFOLDER** function is used to locate the next file or next folder that meets the specified search criteria set by the **FINDFIRSTFOLDER**. The **FINDNEXTFOLDER** function must be used in conjunction with the **FINDFIRSTFOLDER** function.

Return Value

String variable that is passed the name of the folder.

Parameter	Description
searchcriteria	Lets you specify the files or folders for which to search. You can include wild-card characters (* or ?).
attributes	The type of files or folders you want to use. Use the OR operator to specify multiple file and folder types: 1 read-only 2 hidden 4 system 16 Lets you specify to use folders. If not specified, files are used. 32 archive 128 normal (not read-only, hidden, system or archive file or folder) 256 temporary 2048 compressed

Note

- Specifying folder in the **attributes** parameter (16) by itself does not specify a type of folder. You must use another parameter along with 16 to specify a folder type.

Example

```
DIM DCOUNT%,  FCOUNT%             'creates 2 integer variables
DIM FILESARR$(100), DIRARR$(100)  'creates 2 string arrays

REM LOOP #1
REM FIND ALL DIRECTORIES IN THE SAMPLES FOLDER
DCOUNT = 1
DIRARR(DCOUNT) = FINDFIRSTFOLDER("D:\COREL\VENTURA\SAMPLES\*", 16 OR 128)
WHILE (DIRARR(DCOUNT) <> "")
    MESSAGE DIRARR(DCOUNT)
    IF DIRARR(DCOUNT) <> "." AND DIRARR(DCOUNT) <> ".." THEN DCOUNT = DCOUNT + 1
    DIRARR(DCOUNT) = FINDNEXTFOLDER()
WEND

REM LOOP #2
REM FIND ALL *.VP FILES IN EACH DIRECTORY FOUND IN EARLIER LOOP
DIM I%
FCOUNT = 1
FOR I% = 1 TO DCOUNT-1
    FILESARR(FCOUNT) = FINDFIRSTFOLDER("D:\COREL\VENTURA\SAMPLES\" + DIRARR(I%) + "\*.VP", 1 OR
2 OR 4 OR 32 OR 128)
    WHILE (FILESARR(FCOUNT) <> "" )
    MESSAGE DIRARR(I%) & CHR(13) & FILESARR(FCOUNT)
         FCOUNT = FCOUNT + 1
         FILESARR(FCOUNT) = FINDNEXTFOLDER()
    WEND
```

```
NEXT I%
```

In the above example, the first loop fills an array (**DIRARR**) with the names of the normal folders in the D:\COREL\VENTURA\SAMPLES folder. The second loop searches the folders in the SAMPLES folder for any file with the extension VP. Any found VP file has its name added to the **FILESARR** array and has its name displayed in a message box.

The following statement in the first loop is used to remove the current (.) folder and parent (..) folder from being sent to the **DIRARR** array:

```
IF DirArr(Dcount) <> "." AND DirArr(Dcount) <> ".." THEN Dcount = Dcount + 1
```

{button ,AL(`CURRFOLDER;filemode;getfileattr;getcurrfolder;setcurrfolder;',0,"Defaultoverview",)} Related Topics

FREEFILE function

FREEFILE ( )

Returns the lowest file number not associated with an open text file in the computer's memory.

Example

```
OPEN "temp.out" FOR OUTPUT AS 1
OPEN "temp2.out" FOR OUTPUT AS 5
i% = FREEFILE( )
```

The lowest available file number in this example is 2 because 1 is already being used, therefore **i** is set to 2.

{button ,AL(`OPEN_APPEND;OPEN_INPUT;OPEN_OUTPUT;;;',0,"Defaultoverview",)} Related Topics

GETAPPHANDLE function

ReturnValue& = GETAPPHANDLE ( )

Returns the Application Instance Handle for the Corel application that is running a script. For example, if you are running the script from the Corel SCRIPT Editor, GETAPPHANDLE returns the Editor's Application Instance Handle. If you run a script from Corel VENTURA, GETAPPHANDLE returns VENTURA's Application Instance Handle. This function is used in conjunction with DLL calls that require the application's handle.

Return Value

Lets you specify a numeric variable that is passed the Application Instance Handle.

Example

```
hand = GETAPPHANDLE()
```

{button ,AL(`declare_lib;open_output;open_append;open_input;getwinhandle;getapphandle;',0,"Defaultoverview",)}
Related Topics

GETCOMMANDLINE function

ReturnValue$ = GETCOMMANDLINE ( )

Returns the parameters used in the command line that launch a script. To test this command, specify a command line in the Command Line text box in the Corel SCRIPT Editor's Options dialog box (click Tools, Options, Environment tab). For more information about command lines, click ➪.

Return Value

String variable that is passed the command line parameters.

Example

```
CommandLine$ = GETCOMMANDLINE ( )
```

{button ,AL(`setcurrfolder;getCURRFOLDER;getfolder;ht_start_cse_custom;;',0,"Defaultoverview",)} Related Topics

GETCURRFOLDER function

ReturnValue$ = GETCURRFOLDER ( )

Returns the name of the active Windows folder and path.

Return Value

String variable that is passed the name of the active Windows folder and path.

Note

- You can set the active folder using the **SETCURRFOLDER** statement.
- In Corel SCRIPT version 7.0, the **GETCURRFOLDER** function and the **SETCURRFOLDER** statement replace the **CURRFOLDER** statement.

Example

```
SETCURRFOLDER "c:\corel\graphics8\scripts\"
folder$ = GETCURRFOLDER ( )
MyFile$ = "\MyScript.csc"
MyPathFile = folder$ & MyFile&
```

In the above example the first line sets the active folder. The second line assigns the active folder to a string variable. The third line assigns a file name to a string variable. In the last line a string variable is assigned a value which is made by combining the folder and file string variables.

{button ,AL(`setcurrfolder;getCURRFOLDER;getfolder;;;',0,"Defaultoverview",)} Related Topics

GETPROCESSINFO function

ReturnValue& **= GETPROCESSINFO (ProcessHandle)**

This function returns the status of an executable.

Return Value

Numeric variable that is passed a value that indicates whether an executable is running. If the executable is running, this variable is passed the value 259.

Parameter	Description
ProcessHandle	Numeric expression specifying the Windows Process Handle of an executable. Use the **STARTPROCESS** function to determine an executable's Windows Process Handle.

Example

```
launch = STARTPROCESS ("C:\WINDOWS\CALC.EXE")
... 'other script statements
... 'other script statements
... 'other script statements
Calc_Status = GETPROCESSINFO (launch)
```

The above example launches the Windows Calculator and passes the Windows Process Handle to the **launch** variable. The **launch** variable is used with the **GETPROCESSINFO** function to determine whether the Calculator is running.

{button ,AL(`STARTPROCESS ;GETPROCESSINFO;INCLUDE;INPUT;cs_exe_dll;;;',0,"Defaultoverview",)} Related Topics

GETSCRIPTFOLDER function

ReturnValue$ = GETSCRIPTFOLDER ( )

Returns the path and the folder where the executing script resides. If this function is used in a Corel SCRIPT Executable, it returns the path and folder where the Executable resides.

Return Value

String variable that is passed the path and folder of an executing script or Executable.

Note

- If the script has not been previously saved to disk or a network, this function returns an empty string.

Example

```
folder = GETSCRIPTFOLDER ( )
```

{button ,AL(`DECLARE_LIB;OPEN_OUTPUT;OPEN_INPUT;GETWINHANDLE;GETAPPHANDLE;Creating Corel SCRIPT Executables;;;',0,"Defaultoverview",)} Related Topics

GETTEMPFOLDER function

ReturnValue$ = GETTEMPFOLDER ( )

Returns the path and the folder of the system's Windows temporary folder.

Return Value

String variable that is passed the path and folder of the system's Windows temporary folder.

Example

```
t_folder = GETTEMPFOLDER ( )
```

{button ,AL(`GETCURRFOLDER;GETSCRIPTFOLDER;;;;',0,"Defaultoverview",)} **Related Topics**

GETVERSION function

ReturnValue& **= GETVERSION (option)**

Returns the system or Corel SCRIPT version numbers.

Return Value

The GETVERSION function returns one of the following values:

- **Option 0** - Corel SCRIPT run-time interpreter:

  Four-digit number. The first two digits represent the major version number and the last two digits represent the minor version number. For example, the four digit number 7001 indicates major version 7.0 and minor version 01.

- **Option 10** - Corel SCRIPT compiler version:

  Four-digit number. The first two digits represent the major version number and the last two digits represent the minor version number. For example, the four digit number 7001 indicates major version 7.0 and minor version 01.

- **Option 30** - Windows platform:

  Single-digit number indicating the Windows platform that Corel SCRIPT is being used with.

  0 Win32s (Windows 3.11)

  1 Windows 95

  2 Windows NT

- **Option 31** - Windows major version number:

  For example, if you're running Windows 95 and your Windows version number is 4.00.950, the major version number is 4. The same applies to Windows NT.

- **Option 32** - Windows minor version number:

  For example, if you're running Windows 95 and your Windows version number is 4.00.950, the minor version number is 0.

- **Option 33** - Windows build number:

  For example, if you're running Windows 95 and your Windows version number is 4.00.950, the minor version number is 950.

Parameter	Description
option	Lets you specify the system or Corel SCRIPT component to query: 0 Corel SCRIPT run-time interpreter version (the SCINT**xx**.DLL file being used with the current session of Corel SCRIPT) 10 Corel SCRIPT compiler version number 30 Windows platform 31 Windows major version number 32 Windows minor version number 33 Windows build number

Note

- For a script, Executable, DLL, or Corel Add-on created with Corel SCRIPT to run, the major version numbers for Corel SCRIPT (the compiler) and the Corel SCRIPT run-time interpreter (SCINT**xx**.DLL) must be the same or else an error will occur. A difference in the minor version numbers will not cause an error.

Example

```
CS_version = GETVERSION(10)
MESSAGE "Corel SCRIPT major version number " & LEFT (CS_version, 2)
MESSAGE "Corel SCRIPT minor version number " & RIGHT (CS_version, 2)
```

In the above example, the first line passes the Corel SCRIPT version number to **CS_version**. A message box is then used to display the first two digits in CS_version using the **LEFT** function. Next, a message box is used to display the first two digits in CS_version using the **RIGHT** function.

{button ,AL(`Creating Corel SCRIPT Executables;getscriptfolder;getapphandle;getwinhandle;;',0,"Defaultoverview",)}
Related Topics

GETWINHANDLE function

ReturnValue& = GETWINHANDLE ( )

Returns the window handle for the window that is running the script. For example, if you are running the script from the Corel SCRIPT Editor, **GETWINHANDLE** returns the Editor's Windows handle. If you run a script from CorelDRAW, **GETWINHANDLE** returns DRAW's Windows handle. This function is used in conjunction with DLL calls that require the window's handle.

Return Value

Lets you specify a numeric variable that is passed the window's handle.

Note

- If you run a Corel SCRIPT Executable, the **GETWINHANDLE** function returns 0. For more information about Corel SCRIPT Executables, click ⇨.

Example

```
hand = GETWINHANDLE()
```

{button ,AL(`declare_lib;open_output;open_append;open_input;getwinhandle;getapphandle;',0,"Defaultoverview",)}
Related Topics

INCLUDE statement

#INCLUDE filename

Lets you specify a Corel SCRIPT script to execute from the executing script. The specified script is treated as having its contents inserted into the executing script at the line holding the **INCLUDE** statement.

If you re-use many of the same constant, variable, and procedure declarations, you should consider putting that information in a separate script. Keeping this information in a separate script allows you to type the information once, and then call it as many times as you need with an INCLUDE statement in any new script you create.

For example, if you use the **WITHOBJECT** statement to call CorelDRAW 7or Corel VENTURA 7, you can insert the following statements in a INCLUDE file:

```
GLOBAL CONST DRAW7 = "CorelDraw.Automation.7"
GLOBAL CONST VENTURA7 = "CorelVentura.Automation.7"
```

The two statements above create two global constants named **DRAW6** and **VENTURA7**. If you always convert from tenths of a micron to another unit of measurement, you could include the following statement:

```
M_POINT = LENGTHCONVERT (1 , 3 , 1)
```

The **LENGTHCONVERT** statement creates a variable (**M_POINT**) that is equal to the number of tenths of a micron in a point.

Parameter	Description
filename	String expression specifying the filename, and optionally the path. If the path is not specified in **filename**, Corel SCRIPT will search for the file on your system in the following manner: 1 The active folder. You can use the **GETCURRFOLDER** statement to set or determine the active folder. 2 Folders in the path. The path is specified in the systems AUTOEXEC.BAT file. 3 Folders set in the Corel SCRIPT Editor's INCLUDE option. Click ➪ for more information about setting INCLUDE folders. 4 Folders specified with the **ADDFOL** statement.

Note

- Corel SCRIPT 7 introduces and includes CSI files. CSI are scripts (text files) that define commonly used constants. These files can be edited in the Corel SCRIPT Editor, and each Corel application that supports Corel SCRIPT has its own CSI file.

  Based on a typical Corel installation, an application's CSI file resides in the **C:\COREL*CorelSuite**application*\SCRIPTS** folder, where ***CorelSuite*** refers to the Corel products installed and ***application*** refers to the Corel application's folder. For example, the CorelDRAW 7 CSI file may reside in **C:\COREL\DRAW70\DRAW\SCRIPTS** folder and Corel VENTURA 7 CSI file may reside in the **C:\COREL\VENTURA7\VENTURA\SCRIPTS** folder.

  A general constants CSI file (SCPCONST.CSI) for Corel SCRIPT normally resides in the **C:\COREL*CorelSuite*\SCRIPTS** folder.
- The pound sign (#) is required in the syntax.

Example

```
#INCLUDE "My_constants.CSC"
#INCLUDE "SCPCONST.CSI"
```

The above example includes the Corel SCRIPT script MY_CONSTANTS.CSC and the SCPCONST constants file in the executing script.

{button ,AL(`GETCURRFOLDER;lengthconvert;angleconvert;addfol;const;;',0,"Defaultoverview",)} Related Topics

KILL statement

KILL fileName

Deletes a file. This statement is the same as clicking File, Delete in the Windows Explorer or in My Computer in Windows 95.

Parameter	Description
fileName	String expression specifying the filename to delete. You can use wild cards (* and ?) if you want to delete a group of files. For example, **script*.*** deletes all the files in the current folder beginning with **script**. Using **script?.*** deletes all the files in the current folder that begin with **script** and are followed by only one more character.

Note

- An open file cannot be deleted.

Example

```
KILL "temp.out"
```

Deletes the file TEMP.OUT in the current folder.

```
KILL "C:\MyDocs\temp.out"
```

Deletes the file TEMP.OUT in the **C:\MyDocs** folder.

{button ,AL(`rmfolder;open_output;;;;',0,"Defaultoverview",)} Related Topics

MKFOLDER statement and function

Statement: MKFOLDER folderName

Function: ReturnValue& = MKFOLDER (folderName)

Creates a new folder.

Return Value

The MKFOLDER function returns one of the following values:

- TRUE (-1) ⇨ the folder was created
- FALSE (0) ⇨ the folder was not created

Parameter	Description
folderName	String expression specifying the name of the folder to be created. Path information is optional.

Example

```
MKFOLDER "work"
```

Creates the folder **work** as a subfolder of the current folder.

```
success = MKFOLDER ("work")
```

Creates the folder **work** as a subfolder of the current folder and assigns -1 to **success**.

{button ,AL(`;GETCURRFOLDER;SETCURRFOLDER;RMFOLDER;;;;',0,"Defaultoverview",)} Related Topics

REGISTRYQUERY function

ReturnValue = REGISTRYQUERY (MainKey, SubKey, Value)

Returns the value data of a specified value key in the system's Windows registry. This function can help you determine where programs and files are installed on a user's system. This type of information is important when creating scripts that are to run on different system setups.

Return Value

Lets you specify the variable that is passed the value data of a specified value key in the Windows registry. Since this function can pass a string or numeric value, the variable you specify should be a variant. You can use the **GETTYPE** function to determine a variant's subtype.

Parameter	Description
MainKey	Lets you specify the main registry value key to query: 0 HKEY_CLASSES_ROOT 1 HKEY_CURRENT_USER 2 HKEY_LOCAL_MACHINE 3 HKEY_USERS 4 HKEY_PERFORMANCE_DATA 5 HKEY_CURRENT_CONFIG 6 HKEY_DYN_DATA
SubKey	Lets you specify the sub registry value key to query. This must be a complete key path. In Windows 95 for example, "SOFTWARE\Microsoft\Windows\CurrentVersion\Fonts" is a complete path.
Value	Lets you specify the registry value key to query. To specify a default value, use an empty string. Specify an empty string by using two quotation marks ("").

Note

- You cannot use this command to query binary values except those that can be converted to a number.

Example

```
Config_Ventura = REGISTRYQUERY (2, "SOFTWARE\Corel\Corel Ventura\7.0", "ConfigDir")
```

The above example returns the root folder where Corel VENTURA 7 is installed.

```
Arial_file = REGISTRYQUERY (2, "SOFTWARE\Microsoft\Windows\CurrentVersion\Fonts", "Arial
(TrueType)")
```

The above example returns the Arial True Type font's file name.

```
YourName$ = REGISTRYQUERY(2,"SOFTWARE\Corel","UserName")
```

The above example returns the name of the registered owner of Corel Software.

```
CompanyName$ = REGISTRYQUERY(2,"SOFTWARE\Corel","ORGANIZATION")
```

The above example returns the organization name of the registered owner of Corel Software.

```
Phone$ = REGISTRYQUERY(2,"SOFTWARE\Corel","PHONENUMBER")
```

The above example returns the phone number of the registered owner of Corel Software

{button ,AL(`GETSCRIPTFOLDER;GETAPPHANDLE;GETWINHANDLE;GETTYPE;;',0,"Defaultoverview",)} Related Topics

RENAME statement and function

RENAME file_folder1, file_folder2, overwrite

The RENAME statement changes the name of a file or folder, or can be used to move a file. You cannot move a folder using the RENAME statement.

You can also use RENAME as a function: it returns TRUE (-1) if the RENAME operation is successful, FALSE (0) if is not.

Parameter	Description
file_folder1	String expression specifying the name of the file or folder to move. **file_folder1** can include drive and folder path specifics ➪ if path specifics are not included, RENAME assumes the current folder.
file_folder2	String expression specifying the name of the file where **file_folder1** is to be moved. **file_folder2** can include drive and folder path specifics ➪ if path specifics are not included, RENAME assumes the current folder.
overwrite	If **file_folder2** already exists, determines whether to overwrite the existing file (you cannot overwrite existing folders): 0 = rename and overwrite 1 = overwrite fails (default if omitted)

Example

```
' statement example
DIM x AS STRING
DIM y AS STRING
x = "C:\work\example1.vp"
y = "D:\work\example1.vp"
RENAME x, y, 0
```

The above example moves the EXAMPLE1.VP file to the Work folder on the D drive.

```
' statement example
DIM x AS STRING
DIM y AS STRING
x = "C:\work\example1.cdr"
y = "C:\work\example2.cdr"
success = RENAME (x, y, 0)
```

The above example renames the EXAMPLE1.CDR file to EXAMPLE2.CDR, and assigns -1 to **success**.

{button ,AL(`FINDFIRSTFOLDER_FINDNEXTFOLDER;rename;copy;GETCURRFOLDER;rmfolder;SETcurrfolder;;',0,"Defaultoverview",)} Related Topics

RMFOLDER statement and function

RMFOLDER folderName

Removes an existing folder. The folder must be empty before it can be deleted. You can also use **RMFOLDER** as a function: it returns TRUE (-1) if the folder was removed, FALSE (0) if it was not.

Parameter	Description
folderName	String expression specifying the name of the folder to remove. **folderName** can include drive specifics ⇨ if drive specifics are not included, RENAME assumes the current drive.

Example

```
RMFOLDER "C:\TEMP\WORK"
```

Removes the Work folder from the Temp folder.

```
success% = RMFOLDER "C:\TEMP\WORK"
```

Removes the Work folder from the Temp folder and assigns -1 to **success**.

{button ,AL(`KILL;MKFOLDER;GETCURRFOLDER;;;',0,"Defaultoverview",)} Related Topics

SETCURRFOLDER statement

SETCURRFOLDER FolderName

Sets the active Windows folder and path.

Parameter	Description
FolderName	String expression specifying a system folder and path.

Note

- In Corel SCRIPT version 7.0, the **SETCURRFOLDER** statement and the **GETCURRFOLDER** function replace the **CURRFOLDER** statement.

Example

```
SETCURRFOLDER "C:\corel\MyDocs\"
```

The above example sets the active folder and path to C:\COREL\MYDOCS\.

{button ,AL(`GETCURRFOLDER;getfolder;;;',0,"Defaultoverview",)} Related Topics

STARTPROCESS statement and function

Statement: STARTPROCESS exe

Function: ReturnValue& = STARTPROCESS (exe)

This statement or function launches executable files (.EXE files). You can also use this statement to launch Corel applications and Corel SCRIPT Executables.

Return Value

Numeric variable that is assigned a value indicating whether **STARTPROCESS** was not able to launch the specified executable. If the specified executable was not launched, 0 is assigned; otherwise the Windows Process Handle is returned.

Parameter	Description
exe	String expression specifying the executable to launch. The string expression should also include the executable's path and file extension.

Note

- Use the **GETPROCESSINFO** function to determine whether an executable is still running.

Example

```
STARTPROCESS "C:\WINDOWS\CALC.EXE"
```

The above example attempts to launch the Windows Calculator.

```
launch = STARTPROCESS ("C:\WINDOWS\CALC.EXE")
```

The above example attempts to launch the Windows Calculator and assigns a value to the **launch** variable, indicating whether the calculator was launched.

{button ,AL(`STARTPROCESS ;GETPROCESSINFO;INCLUDE;INPUT;cs_exe_dll;;;',0,"Defaultoverview",)} Related Topics

DO...LOOP statements

DO {WHILE | UNTIL} TestCondition
[statements]
LOOP

or

DO
[statements]
LOOP {WHILE | UNTIL} TestCondition

Repeats script instructions while a condition is TRUE or until it becomes TRUE. Note that the first form of syntax may never execute statements if an expression is or is not TRUE, but that the second form always executes statements at least once.

Parameter	Description
TestCondition	Any numeric or string expression that can evaluate to a Boolean (TRUE or FALSE).
[statements]	Series of script instructions to execute and repeat.
{WHILE | UNTIL}	Lets you specify whether to repeat script instructions **WHILE** a condition is TRUE or **UNTIL** it becomes TRUE.

Note

- By placing the condition at the end of the loop, the loop will always be executed once before the condition is tested.
- You can nest **DO...LOOP** statements inside each other up to 20 times.
- You can exit a **DO...LOOP** with the **EXIT** statement.

{button ,AL(`cs_loops;;;;;',0,"Defaultoverview",)} **Related Topics**

Examples for DO...LOOP

```
i% = 5
DO WHILE i% > 0
    i% = i% - 1
LOOP
```

The variable **i%** starts with a value of 5. The loop continues for as long as **i%** has a value greater than 0. Once **i%** equals 0 and the condition (**DO WHILE i% > 0**) is again tested, processing continues at the next statement after the LOOP statement. In this example, the loop executes five times. If the condition were changed to "**i% = 0**", the loop would never execute because the condition would be false immediately.

```
i% = 5
DO UNTIL i% = 0
    i% = i% - 1
LOOP
```

This example is functionally identical to the previous example. The loop executes five times before the condition is TRUE. If the condition were changed to "**i% > 0**", the loop would never execute.

```
i% = 5
DO
    i% = i% - 1
LOOP UNTIL i% = 0
```

By placing the condition at the end of the loop, the loop will always be executed once before the condition is tested. However, in the following example, the loop will still execute five times.

```
i% = 5
DO
    i% = i% - 1
LOOP WHILE i% > 0
```

Again, the loop will execute five times before the condition is no longer TRUE. If the condition were changed to "**i% = 0**", the loop would execute only once before the condition is tested.

{button ,AL(`cs_loops;;;;;',0,"Defaultoverview",)} **Related Topics**

END statement

END { DIALOG | FUNCTION | SELECT | SUB | WITH | WITHOBJECT }

Ends a SELECT CASE, FUNCTION, SUB, WITHOBJECT or DIALOG construct. In Corel SCRIPT 7, the END statement also stops execution of a running Corel SCRIPT script (see the last example). The **STOP** statement can also be used to terminate script execution.

Example

```
BEGIN DIALOG ...
    REM Dialog box statements go here.
END DIALOG

FUNCTION UserFunction%( )
    REM Function processing statements go here.
END FUNCTION

SELECT CASE i%
    CASE 0
        REM Case statements go here.
END SELECT

SUB Subroutine( )
    REM Subroutine statements go here.
END SUB

WITH Dialog1
    REM Dynamic dialog functions go here.
END WITHOBJECT

WITHOBJECT "CorelDraw.Automation.6"
    REM CorelDRAW commands and functions go here.
END WITHOBJECT
```

The script compiler must match a SELECT CASE, FUNCTION, SUB, WITHOBJECT or BEGIN DIALOG statement with a corresponding END statement, as shown in the examples above.

```
IF i% = 3 THEN END
```

If the value of **i%** is 3, then the script stops immediately. No other statements are executed.

{button ,AL(`WITH_Corel_Application;SELECT_CASE;BEGIN_END_DIALOG;BEGIN_END_DIALOG_dyn;with_dialog;SUB_END_SUB;FUNCTION_END_FUNCTION;;',0,"Defaultoverview",)} Related Topics

EXIT statement

EXIT { DO | FOR | FUNCTION | SUB | WHILE}

The EXIT statement is used to exit a procedure or exit a script structure such as a loop.

Parameter	Description
DO	Exits the most tightly enclosed **DO...LOOP** structure. Script execution is sent to the script instruction which follows the **LOOP** statement. If the **EXIT DO** statement is used in a nested loop, script execution is sent to loop nested one level above the loop where the **EXIT** statement is executed.
FOR	Exits the most tightly enclosed **FOR...NEXT** structure. Script execution is sent to the script instruction which follows the **NEXT** statement. If the **EXIT FOR** statement is used in a nested loop, script execution is sent to loop nested one level above the loop where the **EXIT** statement is executed.
FUNCTION	Exits a call to a function. Script execution is sent to the script instruction which follows the call to the function.
SELECT CASE	Exits the most tightly enclosed **SELECT CASE** structure. Script execution is sent to the script instruction which follows the **END SELECT** statement. If the **EXIT SELECT CASE** statement is used in a nested **SELECT CASE** construct, script execution is sent to construct nested one level above the construct where the **EXIT** statement is executed. This option was added in Corel SCRIPT 7.
SUB	Exits a call to a subroutine. Script execution is sent to the script instruction which follows the call to the subroutine.
WHILE	Exits the most tightly enclosed **WHILE...WEND** structure. Script execution is sent to the script instruction which follows the **WEND** statement. If the **EXIT WHILE** statement is used in a nested loop, script execution is sent to loop nested one level above the loop where the **EXIT** statement is executed. This option was added in Corel SCRIPT 7.

Note

- If you do not supply a parameter for the EXIT statement, an exit occurs in the most tightly enclosed structure or procedure.
- Do not confuse the **EXIT** statement with the **END** statement which is used to define the end of a structure or procedure.

Examples

```
FOR i% = 1 TO 5
    IF i = 3 THEN EXIT FOR
NEXT i
```

In the above example, the script execution exits the **FOR** loop when **i** equals 3, not 5.

```
FOR i% = 1 TO 5
    IF i = 3 THEN EXIT FOR
    x% = 5
    DO WHILE x% > 0
        x% = x% - 1
        IF x = 3 THEN EXIT
    LOOP
NEXT i
```

In the above example, the script execution exits the **DO WHILE** loop when **X** equals 3. Since the **EXIT** statement doesn't specify a parameter and the **DO WHILE** loop is the most tightly enclosed structure, the **DO WHILE** is exited.

{button ,AL(`CS_FLOWS;;;;;',0,"Defaultoverview",)} Related Topics

FAIL statement

FAIL ERRNUM

The **FAIL** statement simulates an error and sets the ERRNUM variable to the specified parameter value. **FAIL** can be used to force a call to an error-handling routine. For more information about run-time errors, see Script programming errors.

Parameter	Description
ERRNUM	Any numeric expression. User-defined error numbers are between 800 and 999, inclusive.

{button ,AL(`script_errors;;;;;',0,"Defaultoverview",)} Related Topics

FOR...NEXT statements

FOR counter = start TO end STEP increment
    [statements]
NEXT counter

Repeats (or loops) a group of instructions a specified number of times. The variable loop **counter** has an initial value of **start** and is changed each time the loop statements are executed. The **counter** changes by the value set in the optional **increment** parameter. If the **increment** parameter is not used, **counter** steps, or is increased, by one each time the loop statements are executed.

Parameter	Description
counter	Numeric variable used as counter. Cannot be an array element.
start	Initial value of the counter.
TO	Used to separate the **start** and **end** parameters.
end	Final value of the counter.
STEP	Optional syntax used with the increment parameter.
increment	Amount the counter is incremented each time through loop. If omitted, the default is set to 1. The **increment** parameter can be a positive or negative value. If the **increment** parameter is a positive value, the loop continues to execute as long as **counter** is less than or equal to **end**. If the **increment** parameter is a negative value, the loop continues to execute as long as **counter** is greater than or equal to **end**.
[statements]	Series of script instructions to execute and repeat.

Note

- If the **counter** parameter is omitted in the **NEXT** statement matches to the most recent **FOR** statement.
- If the **start** or **end** parameter is set outside the **counter**'s data type range, an endless loop can occur. For example, if **counter** is an integer and **end** is set to 33000, an endless loop will occur since 33000 beyond the range of an integer. For more information about data types, see Corel SCRIPT data type summary.
- You can nest FOR...NEXT statements inside each other up to 20 times.
- You can exit a FOR...NEXT with the **EXIT** statement.

{button ,AL(`cs_loops;;;;;',0,"Defaultoverview",)} Related Topics

Examples for FOR...NEXT

```
FOR i% = 1 TO 10 STEP 2
    intArray%(i%) = i%
NEXT i%
```

Every other element of the array **intArray%** (elements 1, 3, 5, 7, and 9) are given the value of **i%**.

```
FOR i% = 1 TO UBOUND(stringArray$)
    stringArray$(i%) = "string"
NEXT i%
```

All the elements of the array **stringArray$** are assigned the value "string".

Note

- If you nest **FOR...NEXT**s within **FOR...NEXT**s, you should give each a **counter%** a unique name as shown in the following example:

```
FOR a% = 1 TO 4
    [statements]
    FOR b% = 1 TO 4
        [statements]
        FOR c% = 1 TO 4
            [statements]
        NEXT c%
    NEXT b%
NEXT a%
```

{button ,AL(`cs_loops;;;;;',0,"Defaultoverview",)} **Related Topics**

GOSUB...RETURN statements

GOSUB LabelLine
...
...
LabelLine
...
...
RETURN

The **GOSUB** statement directs script execution to a specified line in the script and then **RETURN** statement returns execution to the script statement following the **GOSUB** statement. You can use the **GOSUB** statement to execute the same block of statements in a script. The **GOSUB** and **RETURN** statements must be within the same procedure in a script.

Parameter	Description
GOSUB LabelLine	Lets you specify the line label or line number to direct script execution to. Line labels cannot be preceded by spaces or tabs and must be followed by a colon. Do not include the colon in the **GOSUB** statement. Line labels can include letters and numbers but must start with a letter. Line numbers cannot be preceded by spaces or tabs and are not followed with a colon. Line numbers cannot include strings or any other non-number characters.

Note

- Consider using user-defined subroutine instead of the **GOSUB** statement.
- Using the **RETURN** statement without a corresponding **GOSUB** statement results in an error.

Example using line label

```
REM Statements...
IF i% = 3 THEN GOSUB BreakOut   'send execution to BreakOut label
...
...
...
IF i% = 4 THEN GOSUB 15         'send execution to line number 15
...
...
...
REM Statements execute here if i% is equal to 3
BreakOut:
    REM More statements execute here.
    RETURN            'execution sent back to line following GOSUB
...
...
...
REM Statements execute here if i% is equal to 4
15
    REM More statements execute here.
    RETURN            'execution sent back to line following GOSUB
```

If **i%** is equal to 3, the statements between the **GOSUB** and the **Breakout** label are skipped. The script processing is sent to the first statement after the label and is then redirected back to the statement that follows the **GOSUB** statement.

If **i%** is equal to 4, the statements between the **GOSUB** and line number 15 are skipped. The script processing is sent to the first statement after the line number and is then redirected back to the statement that follows the **GOSUB** statement.

{button ,AL(`cs_loops;;;;;',0,"Defaultoverview",)} Related Topics

GOTO statement

GOTO { LineLabel | LineNumber }

Directs script execution to a specified line in the script. A **GOTO** statement can branch only to another statement at the same procedure level in the script. For example, you can use a **GOTO** statement in the main section of the script to go to another line in the main section.

Parameter	Description
LineLabel	Lets you specify the line label to go to. The line label cannot be preceded by spaces or tabs and must be followed by a colon. Do not include the colon in the **GOTO** statement. Line labels can include letters and numbers but must start with a letter.
LineNumber	Lets you specify the line number to go to. The line number cannot be preceded by spaces or tabs and is not followed with a colon. Line numbers cannot include strings or any other non-number characters.

Note

- Using **GOTO** statements can make your script difficult to read and debug. You should provide remarks statements when using the **GOTO** statement. Consider using conditional or looping statements instead of the **GOTO** statement.

Example using line label

```
REM Statements...
IF i% = 3 THEN GOTO BreakOut
REM Statements execute here if i% is not equal to 3
...
...
...
REM Statements execute here if i% is equal to 3
BreakOut:
REM More statements execute here.
```

If **i%** is equal to 3, the statements between the **GOTO** and the **Breakout** label are skipped. The script processing is sent to the first statement after the label.

Example using line number

```
REM Statements...
IF i% = 3 THEN GOTO 10
REM Statements execute here if i% is not equal to 3
...
...
...
REM Statements execute here if i% is equal to 3
10
REM More statements execute here.
```

If **i%** is equal to 3, the statements between the **GOTO** and the **10** line number are skipped. The script processing is sent to the first statement after the line number.

Using the **GOTO** statement can easily cause an endless loop in a program. The following example creates an endless loop between the last two statements.

```
REM Statements...
I% = 3
Breakout:
If I% = 3 then GOTO Breakout
```

{button ,AL(`cs_loops;;;;;',0,"Defaultoverview",)} **Related Topics**

IF...THEN...ELSE...ENDIF statements

IF condition1 THEN [statement] ELSE [statement]

or

IF condition1 THEN
[statements]
ELSEIF condition2 THEN
[statements]
ELSEIF condition... THEN
[statements]
ELSE
[statements]
ENDIF

Using the IF statement allows for conditional execution of script instructions based on TRUE or FALSE conditions. If the condition is TRUE, the program performs the THEN statement; if the condition is FALSE, the program performs the ELSE or ELSEIF statement, if included. You can have more than one set of ELSEIF statements in the IF...THEN...ELSEIF...ENDIF construct.

If only one THEN statement is needed, and it is included on the same line as the IF condition, then no ENDIF is required. If the THEN statement is more than one line, then ENDIF is required.

Parameter	Description
condition1, condition2, ...	Any numeric or string expression that can be evaluated as Boolean condition (TRUE or FALSE). If the condition is TRUE, then the statements following the keyword THEN are executed. If the condition is FALSE, then the processing continues at the next ELSEIF statement, if present, or at the statement following the ELSE keyword, if present.
[statement]	Script instruction to execute.
[statements]	Series of script instructions to execute.

Note

- You can nest IF statements inside each other up to 20 times.

{button ,AL(`cs_loops;select_case;;;;',0,"Defaultoverview",)} **Related Topics**

Examples for IF...THEN...ELSE...ENDIF

```
IF i% = 0 THEN MESSAGE "The variable is 0."
```

If the variable **i%** has a value of 0, a message dialog box appears.

```
IF i% = 0 THEN MESSAGE "The variable is 0." ELSE BEEP
```

If the variable **i%** has a value of 0, a message appears, otherwise a beep is sounded. Because the entire IF statement is on one line, no ENDIF is needed.

```
IF i% = 0 THEN
    MESSAGE "The variable is 0"
    i% = 1
ELSEIF i% = 1 THEN
    MESSAGE "The variable is less than 2"
ELSEIF i% = 2 THEN
    MESSAGE "The variable is greater than 1"
ELSE
    BEEP
ENDIF
```

You must use the multi-line **IF...THEN...ENDIF** when there is a second condition to test after the first IF or when there is more than one statement to process as a result of the condition. Repeat the **ELSEIF** statements for as many conditions as needed. If many conditions must be tested, you might want to use the SELECT CASE statement. Note that in the example, even though **i%** is assigned a value of 1 in the **i%** = 0 condition, the **ELSEIF i%** = 1 condition does not execute. Once a condition is TRUE, processing continues after the **ENDIF** statement once the statements that meet the condition execute.

You can evaluate variables without assigning results. For example:

```
IF (abc<=15.3) then BEEP ELSE MESSAGE "It's greater than 15.3"
```

{button ,AL(`cs_loops;select_case;;;;',0,"Defaultoverview",)} **Related Topics**

ON ERROR statement

ON ERROR { GOTO line | RESUME NEXT | EXIT }

The **ON ERROR** statement sets an error-handling routine. The error-handling routine is a series of instructions within the same procedure that are executed when an error occurs. If you don't use an error-handling routine, a run-time error stops script execution. For more information about run-time errors, see Script programming errors.

When an error occurs, an error value is passed, or trapped, to the Corel SCRIPT global variable ERRNUM. For example, if a division by zero error occurs, ERRNUM equals 100. For more information, see Error Codes.

Parameter	Description
GOTO line	Script execution is directed to a specified line when an error occurs. The line must be in the same procedure where the error occurred. See the **GOTO** statement for more information about the **line** parameter.
RESUME NEXT	Script execution continues to the line immediately following the line where the error occurred.
EXIT	Disables the **ON ERROR** setting in the current procedure.

Error-handling routines

An error-handling routine is not a separate procedure but a block of script instructions within the same procedure as the **ON ERROR** statement. You can use the following syntax in the error-handling routine to return execution to a procedure from an error-handling routine:

Parameter to return execution	Description
RESUME	Placing a **RESUME** statement at the end of an error-handling routine re-executes the script instruction which caused the error. If you use the RESUME statement, you should ensure that your error-handling routine resolves the error, otherwise an infinite loop is likely to occur.
RESUME NEXT	Placing a **RESUME NEXT** statement at the end of an error-handling routine directs script execution to the line immediately following the line where the error occurred.
RESUME AT **line**	Placing a RESUME AT statement at the end of the error-handling routine directs script execution to the specified line. The line must be in the same procedure where the error occurred. You can use line labels or line numbers to specify a line. See the **GOTO** statement for more information about the **line** parameter.

Note

- You should place a **STOP**, **EXIT FUNCTION**, or an **EXIT SUB** before an error-handling routine in a script to prevent it from being executed when no error has occurred.
- If an error occurs in an error-handling routine, script execution is redirected to the script instruction that initiated the error-handling routine.
- Not all Corel SCRIPT errors are trappable.
- If an error occurs in an error handler, control returns to the calling procedure. If the calling procedure has error handling, it is activated to handle the error. If the calling procedure's error handler is also active, control passes back to the first calling procedure that is enabled, and inactive.
  - If no inactive, enabled error handler is found, the error terminates the script. Each time the error handler passes control back to the calling procedure, that procedure becomes the current procedure. Once an error is handled by an error handler in any procedure, execution resumes in the current procedure at the point designated by the **RESUME** statement.
  - The **ERRNUM** value corresponds to the cause of the most recent error. An error handler should test or save the **ERRNUM** value before another error can occur, or before a procedure that could cause an error is called.

{button ,AL(`script_errors;;;;;',0,"Defaultoverview",)} Related Topics

Example for ON ERROR and FAIL

```
DECLARE FUNCTION asknum$()
ON ERROR GOTO mainerrhan

num$=asknum$()
MESSAGE "You entered "+num$
STOP

' main error handling routine
mainerrhan:
    SELECT CASE ERRNUM
    CASE 800
        MESSAGE "Please enter a higher number"
        RESUME
    CASE 801
        MESSAGE "Please enter a lower number"
        RESUME
    CASE ELSE
        MESSAGE "Unexpected error"
        RESUME NEXT
    END SELECT
STOP

' This function will ask for a number between 10 and 20.
FUNCTION asknum$
    ON ERROR GOTO asknumerror
    number$=INPUTBOX("Please enter a number between 10 and 20")
    somenum& = VAL(number$)
    IF somenum&<10 THEN FAIL 800
    IF somenum&>20 THEN FAIL 801
    asknum$=number$
    EXIT FUNCTION

    askNumError:
        ' Error handling is passed to the main section
END FUNCTION
```

In the above example, a function is used to create an input box that takes a number. If the number input is not between 10 and 20 inclusive, an error-handling routine is called. (The **FAIL** statement is used to simulate an error number.) The error-handling routine displays a message and then returns to the input box again.

{button ,AL(`script_errors;;;;;',0,"Defaultoverview",)} Related Topics

REM statement

REM comment

' comment

A non-executed remark, or comment, in a program. You can use an apostrophe (') instead of REM. Everything from REM (or the apostrophe) to the end of the line is a remark. The REM statement must be placed at the beginning of a line. The apostrophe can be placed anywhere in a line.

If a script's first line, second line, or both are REM statements, the comments are displayed in a Corel application's Run Script dialog box if the script is specified. The same two REM statements are also displayed in the status bar if the script is assigned to a tool bar button.

Parameter	Description
comment	Any text you want to include in your script as a comment.

Example

```
REM Description: This is a description.
REM Created by: Your Name
' This first statement sets up the variables
DIM int2Array%(10) ' The rest of this line is a comment
' More processing follows here.
```

{button ,AL(`Using_flow_control_statements;To_add_a_button_to_a_toolbar;;;;',0,"Defaultoverview",)} Related Topics

SELECT CASE...END SELECT statements

SELECT CASE testexpression
    CASE {caseexpression | caseexpression TO caseexpression | IS reloperator caseexpression} , ...
        [statements]
    CASE {caseexpression | caseexpression TO caseexpression | IS reloperator caseexpression} , ...
        [statements]
    CASE ELSE
        [statements]
END SELECT

Compares the value of a test expression to the values in the **CASE** statements and executes blocks of statements dependent on their relationships. If none of the values in the **CASE** statements match the test expression, the **CASE ELSE** statements are executed.

Parameter	Description
testexpression	Any numeric or string expression.
caseexpression	An expression that evaluates to the same data type as **testexpression**. You can have more than one set of CASE statements in the SELECT CASE construct.
reloperator	A relational operator such as =, <>, >, <, >=, or <=.
[statements]	Program instructions conditionally executed; any number of statements on one or more lines.
TO	Corel SCRIPT keyword used to specify a range of values. The smaller value must precede TO.
IS	Corel SCRIPT keyword used with relational operators to specify a range of values.

Note

- You can exit a **SELECT CASE** with the **EXIT** statement. This is new for Corel SCRIPT 7.0.

{button ,AL(`IF_THEN_ELSE_ENDIF;Relational_Operators;;;;',0,"Defaultoverview",`main')} Related Topics

Example for SELECT CASE

```
SELECT CASE i%
    CASE -10 TO -5, 5 TO 9, IS = 10
        MESSAGE "Between -10 and -5 or 5 and 10"
    CASE -4 TO -1, 2 TO 4
        MESSAGE "Between -4 and 4 but not 0 or 1"
    CASE 0, 1
        MESSAGE "Zero or one"
    CASE ELSE
        MESSAGE "Greater than 10 or less than -10"
END SELECT
```

In the above example:

- If **i%** is between -10 and -5 or 10 and 5, the message "Between -10 and -5 or 5 and 10" appears.
- If **i%** is between -4 and 4, but is not zero or one, the message "Between -4 and 4 but not 0 or 1" appears.
- If **i%** is zero or 1, the message "Zero or one" appears.
- If **i%** is not any of those numbers, the message "Greater than 10 or less than -10" appears.

{button ,AL(`cs_loops;;;;;',0,"Defaultoverview",)} **Related Topics**

STOP statement

STOP

Stops execution of a running Corel SCRIPT script. The **END** statement can also be used to terminate script execution.

Example

```
IF i% = 3 THEN STOP
```

If the value of **i%** is 3, then the script stops immediately. No other statements are executed.

The STOP statement is often used to stop a running script when a Cancel button is pressed in a dialog box as shown in the following example:

```
BEGIN DIALOG Buttons1 55, 34, 236, 40, "BUTTON example"
    OKBUTTON 21, 12, 40, 14
    CANCELBUTTON 71, 12, 40, 14
    PUSHBUTTON 121, 12, 40, 14, "&Push"
    HELPBUTTON 171, 12, 40, 14, "C:\Help.hlp", 1044
END DIALOG
ret = DIALOG(Buttons1)
IF ret = 1 THEN MESSAGE "OK button chosen"
IF ret = 2 THEN STOP    'Cancel pressed
IF ret = 3 THEN MESSAGE "Push button chosen"
```

{button ,AL(`END;;;;;',0,"Defaultoverview",`main')} Related Topics

WHILE...WEND statements

WHILE TestCondition
[statements]
WEND

Continuously executes (or loops) script instructions as long as a test condition is TRUE.

Parameter	Description
TestCondition	Any numeric or string expression that can evaluate to a Boolean (TRUE or FALSE).
[statements]	Series of script instructions to execute.

Note

- You can nest **WHILE...WEND** statements inside each other up to 20 times.
- You can exit a **WHILE...WEND** with the **EXIT** statement. This is new for Corel SCRIPT 7.0.

Example

```
i% = 5
WHILE i% > 0
    i% = i% - 1
WEND
```

The loop executes for as long as the condition **i%** is greater than 0 is TRUE. In this example, the loop will execute five times.

{button ,AL(`cs_loops;;;;;',0,"Defaultoverview",)} Related Topics

WITHOBJECT...END WITHOBJECT statements

WITHOBJECT application
[statements]
END WITHOBJECT

The **WITHOBJECT** construct enables you to send commands to applications which support OLE Automation with an automation object. This command can be used to send commands to both Corel and non-Corel applications. Click ⇨ for more information about OLE automation.

Parameter	Description
application	A string expression specifying the application to call. Click ⇨ for a list of Corel applications that support Corel SCRIPT and application strings.

[statements] Series of script instructions to execute. The instructions can be a combination of both Corel SCRIPT application commands, Corel SCRIPT programming statements, and non-Corel application commands and functions.

Note

- You can nest **WITHOBJECT...END WITHOBJECT** statements within other **WITHOBJECT...END WITHOBJECT** statement blocks.
- Once your script is in **WITHOBJECT...END WITHOBJECT** statement block you cannot change the object you are sending instructions to without using another set of **WITHOBJECT...END WITHOBJECT** statements.
- If you have more than one session of your application open, the script is executed in the application which was opened first.
- The application that the **WITHOBJECT** statement is calling does not have to be opened to be called.
- In Corel SCRIPT 6, predefined constants were used to specify an application. For example **DRAW** for **CorelDraw.Automation.6**. This option no longer exists but you can now use constants defined in the **INCLUDE** files distributed with script. See the **INCLUDE** statement for more information.

Example

```
WITHOBJECT "CorelDraw.Automation.6"
    .InsertPages -1, 2
    .SetPaperColor 2, 0, 255, 0, 0
END WITHOBJECT
```

This example inserts two pages into a CorelDRAW document and changes the paper color.

{button ,AL(`ole_cs;end;corel_script_programming_language;Executing_script_files;Corel_SCRIPT_advanced_programming_features;;',0,"Defaultoverview",)} Related Topics

To add a dialog box to a script using the Corel SCRIPT Editor

1. Place the insertion point in the script line before where you want to insert the dialog box definition.
2. Click Tools, Dialog.

   The **BEGIN DIALOG** and **END DIALOG** statements are inserted into the active script.

   A dialog window also opens containing an empty dialog box that can be edited. As you edit the dialog box, the script that launched the dialog window reflects the changes you make to the dialog box. For example, if you add a check box to the dialog box, a CHECKBOX statement is added to the script.

Note

- While the dialog window for the new dialog box is open you cannot edit it's dialog definition in the script window.
- By default, all new dialog boxes are of the static type. See To edit a dialog box's attributes for more information.

{button ,AL(`a_trans;;;;;',0,"Defaultoverview",)} Related Topics

To edit dialog box definition in a script using the Corel SCRIPT Editor

1. Place the insertion point in the a script line containing the dialog box definition you want to edit.
2. Click Tools, Dialog.

   A dialog window opens containing a dialog based on the dialog definition statements from the **BEGIN DIALOG** statement to the **END DIALOG** statement.

   While the dialog window for the dialog box is open you cannot edit it's dialog definition in the script window.

Note

- If the dialog definition statements you want to edit contain invalid dialog definition statements, you're prompted to either ignore the statement that contains the invalid dialog definition statement or create a new dialog box.

{button ,AL(`a_trans;;;;;',0,"Defaultoverview",)} Related Topics

To close a Corel SCRIPT dialog window

- Click File, Close.

Note

- To use this command, the dialog window you want to close must be active.
- Once the dialog window is closed, the cursor returns to script window that launched it.

{button ,AL(`a_start;;;;;',0,"Defaultoverview",)} **Related Topics**

To save a dialog box as a separate script file

- Click File, Export Dialog. Type a name in the File name box.

  Dialog boxes are saved as Corel SCRIPT statements. The statements form a dialog box definition which starts with the BEGIN DIALOG statement, ends with the END DIALOG statement, and has dialog control statements in between.

Note

- You can only issue this command if a dialog window is active.

{button ,AL(`ht_file_menu_cse;script_files;;;;',0,"Defaultoverview",)} Related Topics

To copy a dialog box definition from a dialog window to a script window

From a dialog window:

1. Click Edit, Select All.
2. Click Edit, Copy.

   The dialog box definition is transferred to the Clipboard as Corel SCRIPT statements consisting of the BEGIN DIALOG and END DIALOG statements with the control statements in between.
3. In a script window, place the insertion point where you want to insert the dialog box definition.
4. Click Edit, Paste. Selected text in the Editor is overwritten with the Clipboard contents.

Note

- You can also transfer dialog control statements from a dialog window to a script window. Cut or copy the selected controls to the Clipboard and then paste them into a script window.

{button ,AL(`a_trans;a_insert;a_start;ht_start_cse_app;ht_start_cse_win;;',0,"Defaultoverview",)} Related Topics

To move or copy a dialog control script statement to a dialog window

From a script window:

1. Select dialog control statements.
2. Click Edit, Cut to move or Edit, Copy to copy. The dialog control statements are place on the Clipboard.
3. Click in the dialog box in the dialog window.
4. Click Edit, Paste.

{button ,AL(`a_trans;a_insert;a_start;ht_start_cse_app;ht_start_cse_win;;',0,"Defaultoverview",)} Related Topics

To edit a dialog box's attributes

1. Select the dialog box.
2. Click Edit, Attributes.
3. Type a new value into any of the following attribute boxes:
   - Width (in dialog units)
   - Height (in dialog units)
   - X (left border position)
   - Y (top border position)
   - Title
   - Dialog Name
   - Dialog Function (used with dynamic dialogs)
   - Comment (a remark is added to the BEGIN DIALOG statement).
4. Enable or disable the Center Dialog option and the Dynamic Dialog Mode checkbox.

Note

- You can convert a static dialog box to a dynamic dialog box by enabling the Dynamic Dialog Mode checkbox. Disabling the checkbox converts a dynamic dialog box to a static dialog box.
- If the Center Dialog option is enabled, the X and Y number boxes are grayed and cannot be edited. By default, a new dialog has the Center Dialog checkbox enabled.

  A thick border around a dialog box indicates it has been selected. Dialog boxes don't use sizing handles.

{button ,AL(`a_edit;htde_select_dialog_box;;;;',0,"Defaultoverview",)} Related Topics

To resize a dialog box using the mouse

1. Place the mouse pointer on a dialog border. The mouse pointer changes shape.
2. Drag the side, top or bottom border to resize the window in one direction or drag a border corner to resize the window horizontally and vertically.
3. Release the mouse button when the window is the desired size.

Note

- Pressing ESC while resizing resets the dialog box to its original size.

{button ,AL(`a_edit;;;;;',0,"Defaultoverview",)} Related Topics

To move a dialog box using the mouse

1. Position the mouse pointer on the dialog box title bar.
2. Hold the mouse button down and drag the dialog box to a new position.

Note

- The dialog window is a representation of your computer screen. When you move the dialog box within the dialog window you are actually changing the dialog box's screen placement when it is run in a script. For example, if you move the dialog box to the bottom-right corner of its dialog window, it will appear on the bottom-right corner of the computer screen when run in a script file.
- If the Center Dialog checkbox in the dialog attributes box is enabled, the dialog box position in the dialog window is ignored and the dialog box is centered during script running. A dialog box can also be set to be centered on the screen by omitting the position attributes parameters in the **BEGIN DIALOG** statement.
- By default, a new dialog has the Center Dialog checkbox enabled.

{button ,AL(`a_edit;;;;;',0,"Defaultoverview",)} **Related Topics**

To insert a control into a dialog box using the click method

1. Click Control and then click the control you want to insert.
2. Position the mouse pointer in the dialog box.

   The mouse appears in the Control state (⇨) when positioned in a dialog window.
3. Click in the dialog box where you want to place the control's top-left corner.

   The control is inserted with its default size settings.

Note

- To insert a control a multiple number of times, hold down CTRL while you make your control selection. Press ESC to return the mouse back to the Selector state (⇨).

{button ,AL(`a_edit;htde_default_sizes;a_select;;;',0,"Defaultoverview",)} Related Topics

To insert a control into a dialog box using the click & drag method

1. Click Control and then click the control you want to insert.
2. Position the mouse pointer in the dialog box.

   The mouse appears in the Control state (⇨) when positioned in a dialog window.
3. Position the mouse pointer where you want one corner of the control to appear
4. Hold the mouse button down and drag up or down on a diagonal.
5. When the control is the size and shape you want, release the mouse button.

Note

- To insert a control a multiple number of times, hold down CTRL while you make your control selection. Press ESC to return the mouse back to the Selector state (⇨).
- The control is inserted with a default label and identifier which you can change.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To delete controls

1. Select the controls you want to delete.
2. Click Edit, Delete.

   The selected controls are not transferred from the dialog box to the Clipboard.

Note

- Instead of using Edit, Delete, you can delete controls by clicking Edit, Cut which transfers controls from the dialog box to the Clipboard as Corel SCRIPT statements.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To cut controls to the Clipboard

1. Select the controls you want to cut.
2. Click Edit, Cut.

   The selected controls are transferred from the dialog box to the Clipboard as Corel SCRIPT statements.

Note

- Instead of using Edit, Cut, you can delete controls by clicking Edit, Delete. This does not transfer controls from the dialog box to the Clipboard.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To copy controls to the Clipboard

1. Select the controls you want to copy.
2. Click Edit, Copy.

   The selected controls are copied from the dialog box to the Clipboard as Corel SCRIPT statements.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To paste a copy of a control in a dialog box

1. Select the control(s) you want to copy.
2. Click Edit, Copy.

   The selected control(s) is copied from the dialog box to the Clipboard as Corel SCRIPT statements.
3. Click in the dialog window you want to paste the control(s) into.
4. Click Edit, Paste.

Note

- If you try to paste Clipboard contents into a dialog window that contains invalid dialog definition statements, you're prompted to either ignore the statement that contains the invalid dialog definition statement or create a new dialog box.
- A pasted control retains the original's label, identifier, size, and position attributes.
- If you paste a control into the same dialog window it was copied from, the pasted control is placed on top of the original control with the same name and identifier. You'll have to change the identifier in one of the controls because identifiers must be unique in a dialog definition.
- If you try to paste controls from a different sized dialog box into a position that doesn't exist in the second dialog box, the controls are placed in the closest valid position.
- If the controls are too big to fit into the second dialog box, the controls are resized to fit.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To move a control in a dialog box using the mouse

1. Select the control(s) you want to move.
2. Press and hold the mouse button.
3. Drag the control(s) to a new location.
4. Release the mouse button.

Note

- As you are moving the controls, the Properties Bar in the dialog window shows the control coordinates of the last control you selected. The last control you select has a dotted line border.
  - If Snap to Grid is enabled, the control moves along the dialog box grid.
  - If Snap to Grid is enabled and more than one control is selected, the last control you selected moves along the dialog box grid.
  - Pressing ESC while moving the control(s) resets the control(s) to its original position.

Tip

You can also move selected controls by using the arrow keys on the keyboard.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To move a control in a dialog box using the attributes box

1. Select the control(s) you want to move.
2. Click Edit, Attributes.
3. Type a number in the X number box for the distance from the inside of the dialog box's left border to the control's left border. The number is based in dialog units.
4. Type a number in the Y number box for the distance from the bottom of the dialog box's title bar to the control's top border. The number is based in dialog units.

Notes

- If more than one control is selected, the X and Y number boxes are cleared because they cannot show a mixed-value. Though cleared, the number boxes accept entries. Entering values in the X and Y number boxes positions the top-left corners of the selected controls on the on the same X-Y coordinate in the dialog box.
- You can also move and resize controls by using the Properties bar in a dialog window. Click View, Properties bar to first display the bar. Select a control(s) and type the new coordinates or size values in the number boxes on the properties bar.

Tip

- You can also move selected controls by using the arrow keys on the keyboard.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To move a control from one dialog box to another

1. Select the control(s) you want to move.
2. Click Edit, Cut.
3. By clicking in it, activate the dialog window that you want to move the control(s) into.
4. Click Edit, Paste.

   The control(s) are placed in the same position as in the first dialog box.

Notes

- If you try to move controls from a different sized dialog box into a position that doesn't exist in the second dialog box, the controls are placed in the closest valid position.
- If the controls are too big to fit into the second dialog box, the controls are resized to fit.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To resize a control in a dialog box using the mouse

1. Select the control(s) you want to resize.
2. Press and hold the mouse button on a sizing handle.

   The sizing handles on the corners change both the width and height. The other sizing handles change either the width or height.
3. Drag the handle until the control is the size you want.
4. Release the mouse button.

Notes

- As you are resizing a control, the Properties Bar in the dialog window shows the control's new size and coordinates. If more than one control is selected, the Properties Bar in the dialog window shows the size of the last control you selected.
- If Snap to Grid is enabled, the control is resized along the dialog box grid.
- If Snap to Grid is enabled and more than one control is selected, the last control you selected is resized along the dialog box grid.
- Pressing ESC while resizing resets the control(s) to its original size.
- You can also move and resize controls by using the Properties bar in a dialog window. Click View, Properties bar to first display the bar. Select a control(s) and type the new coordinates or size values in the number boxes on the properties bar.

Tip

You can also resize selected controls by holding down the SHFIT key and using the arrow keys on the keyboard.

{button ,AL(`a_edit;htde_control_size_label;a_select;;;',0,"Defaultoverview",)} **Related Topics**

To resize a control in a dialog box using the attributes box

1. Select the control(s) you want to resize.
2. Click Edit, Attributes.
3. Type a new value in the Width number box to change the width. The value is expressed in dialog units.
4. Type a new value in the Height number box to change the height. The value is expressed in dialog units.

Notes

- Steps 3 and 4 are both optional. You can do one or both.
- If more than one control is selected, they are resized to the same width and height as specified in the Multiple Selection Attributes dialog box.
- You can also move and resize controls by using the Properties bar in a dialog window. Click View, Properties bar to first display the bar. Select a control(s) and type the new coordinates or size values in the number boxes on the properties bar.

Tip

You can also resize selected controls by holding down the SHFIT key and using the arrow keys on the keyboard.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To edit a control's attributes using the attributes box

1. Select the control(s) you want to edit.
2. Click Edit, Attributes.
3. Type a new value into any of the following attribute boxes, if applicable (this is not a complete list of control attributes but some of the most common):
   - Width
   - Height
   - X (left border position)
   - Y (top border position)
   - Text
   - Value
   - Option Group
   - Comment (a remark is added to the Corel SCRIPT statement for the control).

Notes

- If more than one control is selected, the selected controls take on the attributes specified in the Multiple Selection Attributes dialog box.
- You can also move and resize controls by using the Properties bar in a dialog window. Click View, Properties bar to first display the bar. Select a control(s) and type the new coordinates or size values in the number boxes on the properties bar.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} Related Topics

To duplicate a control

1. Select the control(s) you want to duplicate.
2. Click Edit, Duplicate.

Notes

- The duplicated control(s) is offset from the original by 3 dialog units, both down and to the right.
- The duplicated control(s) takes on the default identifier and the original control's label. If you copied the control, the original's label and identifier would also be copied.

{button ,AL(`a_edit;a_select;;;;',0,"Defaultoverview",)} **Related Topics**

To test a dialog box

- Click View, Test Dialog.

  In test mode you can confirm the following dialog box features:

- tab order within the dialog box
- shortcut keys are operational
- drop-down boxes openings

Notes

- Press ESC to exit test mode. Pressing any push button or the Close Dialog button () also exits test mode.
- You cannot edit a dialog box in test mode.
- The following controls are filled with place holders in test mode:
  - list boxes
  - drop-down list boxes
  - combo boxes
  - drop-down combo boxes
  - image boxes
  - bitmap buttons
  - spin controls
- The place holders give you a better idea of what the dialog box will actually look like when it is run rather than an empty control.

{button ,AL(`a_focus;;;;;',0,"Defaultoverview",)} **Related Topics**

To select a control

1. Click Control, Selector. By default, the mouse is in Selector mode and appears as  .
2. In the dialog box, click anywhere on a control.

Note

- A selected control has a 8 sizing handles.

{button ,AL(`a_select;;;;;',0,"Defaultoverview",)} **Related Topics**

To select a dialog box

- Click anywhere in a dialog box until a thick border is displayed around it.

Note

- A thick border around a dialog box indicates it has been selected. Dialog boxes don't use sizing handles.

{button ,AL(`a_select;htde_dialog_attributes;;;;',0,"Defaultoverview",)} **Related Topics**

To select multiple controls

1. Click the Control, Selector. By default, the mouse is in Selector mode and appears as  .
2. In the dialog document window, while holding down the SHIFT key, click the controls you want to select.

Notes

- Selecting more than one control lets you apply the same attributes to each of them.
- A selected control has a 8 sizing handles.
- You can also select multiple controls using a marquee select.
- The last control you select has a dotted line border.

{button ,AL(`a_select;;;;;',0,"Defaultoverview",)} Related Topics

To marquee select controls

1. Click the Control, Selector. By default, the mouse is in Selector mode and appears as  .
2. Hold down the mouse button and drag the marquee box until it completely encloses the controls you want selected.
3. Release the mouse button.

Note

- When you hold down ALT while you drag, any control that intersects with the marquee box is selected. If you do not release the mouse button before releasing ALT, only those controls enclosed by the marquee box will be selected.
- A selected control has a 8 sizing handles.
- The last control you select has a dotted line border.

{button ,AL(`a_select;;;;;',0,"Defaultoverview",)} **Related Topics**

To add or remove controls to a group of selected controls

1. Click the Control, Selector. By default, the mouse is in Selector mode and appears as  .
2. In the dialog document window, while holding down the SHIFT key, click the controls you want to select or de-select.

Notes

- A selected control has a 8 sizing handles.
- The last control you select has a dotted line border.

{button ,AL(`a_select;;;;;',0,"Defaultoverview",)} **Related Topics**

To deselect all controls

- Click any open space outside the dialog box in the dialog window.

{button ,AL(`a_select;;;;;',0,"Defaultoverview",)} **Related Topics**

To select all controls in a dialog

- Click Edit, Select All.

Tips

- Selecting the dialog box selects all the controls in the dialog as well.
- Selecting all the controls in a dialog also selects the dialog box.
- The last control you select has a dotted line border.

{button ,AL(`a_select;;;;;',0,"Defaultoverview",)} Related Topics

To align controls along their left edge

1. Select the controls you want to align.
   The last control you select maintains its position; all other selected controls move to align with this control.
2. Click Layout, Align Control, Left.

Note

- The last control you select has a dotted line border.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To align controls along their right edge

1. Select the controls you want to align.
   The last control you select maintains its position; all other selected controls move to align with this control.
2. Click Layout, Align Control, Right.

Note

- The last control you select has a dotted line border.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To align controls along their top edge

1. Select the controls you want to align.
   The last control you select maintains its position; all other selected controls move to align with this control.
2. Click Layout, Align Control, Top.

Note

- The last control you select has a dotted line border.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To align controls along their bottom edge

1. Select the controls you want to align.
   The last control you select maintains its position; all other selected controls move to align with this control.
2. Click Layout, Align Control, Bottom.

Note

- The last control you select has a dotted line border.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To make controls the same width

1. Select the controls you want to resize.
   The last control you select maintains its width; all other selected controls resize to this control.
2. Click Layout, Make Same Size, Width.

Note

- The last control you select has a dotted line border.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To make controls the same height

1. Select the controls you want to resize.
   The last control you select maintains its height; all other selected controls resize to this control.
2. Click Layout, Make Same Size, Height.

Note

- The last control you select has a dotted line border.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To make controls the same width and height

1. Select the controls you want to resize.
   The last control you select maintains its size; all other selected controls resize to this control.
2. Click Layout, Make Same Size, Both.

Note

- The last control you select has a dotted line border.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To size a control to fit its label

1. Select the control(s) you want to resize.
2. Click Layout, Size to Content.

Note

- You can use this command on option buttons, check boxes, push buttons, and text.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To center controls vertically in a dialog box

1. Select the controls you want to center.
2. Click Layout, Center in Dialog, Vertical.

   The selected controls are centered vertically based on the topmost and bottommost borders of the selected controls.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To center controls horizontally in a dialog box

1. Select the controls you want to center.
2. Click Layout, Center in Dialog, Horizontal.

   The selected controls are centered horizontally based on the leftmost and rightmost borders of the selected controls.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} **Related Topics**

To even the spacing between controls horizontally within a dialog box

1. Select the controls you want to arrange.
2. Click Layout, Distribute, Horizontal.

   The selected controls are spaced evenly between the leftmost and rightmost borders of the selected controls.

Note

- More than two controls must be selected for this option to be enabled.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} Related Topics

To even the spacing between controls vertically within a dialog box

1. Select the controls you want to arrange.
2. Click Layout, Distribute, Vertical.

   The selected controls are spaced evenly between the topmost and bottommost borders of the selected controls.

Note

- More than two controls must be selected for this option to be enabled.

{button ,AL(`a_align;;;;;',0,"Defaultoverview",)} **Related Topics**

To turn on or turn off Snap to Grid

- Click Layout, Snap to Grid.

Note

- Repeat the above procedure to turn off Snap to Grid
- When Snap to Grid is enabled, a control can only be moved along the dialog box grid. Snap to Grid can be enabled without showing the grid.
- Snap to Grid is set for all dialog windows.

{button ,AL(`a_edit;a_snap;a_insert;a_align;;',0,"Defaultoverview",)} **Related Topics**

To view or hide the grid

1. Click Tools, Grid Settings.
2. Enable Show Grid

   Show Grid is enabled when a check mark appears beside Show Grid.

Note

- Click Show Grid again to hide the grid. Show Grid is disabled when no check mark appears beside Show Grid.
- When Snap to Grid is enabled, a control can only be moved along the dialog box grid. Snap to Grid can be enabled without showing the grid.
- Show Grid is set for all dialog windows.

{button ,AL(`a_edit;a_snap;a_insert;a_align;;',0,"Defaultoverview",)} Related Topics

To view or hide a dialog window's Properties bar

- Click View, Properties Bar.

  A check mark beside the Properties Bar menu command indicates the Properties bar is displayed in the active dialog window.

Note

- You can only issue this command if a dialog window is active.

{button ,AL(`ht_view_wins;;;;;',0,"Defaultoverview",)} **Related Topics**

To set grid spacing

1. Click Tools, Grid Settings.
2. Type a new value in the Horizontal number box to set the horizontal spacing.
3. Type a new value in the Vertical number box to set the vertical spacing.

Notes

- Grid measurements are expressed in dialog units.
- To show the grid, enable Show Grid in the same dialog box.
- Grid spacing settings are set for all dialog windows.

{button ,AL(`a_edit;a_snap;a_insert;a_align;;',0,"Defaultoverview",)} **Related Topics**

To add a shortcut key to a control that has a label

1. Select a control that has a label (option buttons, push buttons, check boxes).
2. Click Edit, Attributes.
3. In the Label text box place an ampersand (&) in front of the letter you want to use as a shortcut key.

Note

- A shortcut key is an alternative to using TAB to move within a dialog and change focus. To use a shortcut, press ALT+***!*** where ***!*** is the defined shortcut key.
- Shortcut keys in a dialog box should be unique.

{button ,AL(`a_focus;;;;;',0,"Defaultoverview",)} **Related Topics**

To add a shortcut key to a control that doesn't have a label

The text control that precedes a control without a label (for example, list boxes and text boxes) can be used to hold the shortcut for the control. The text control statement must immediately precede the control's statement in the dialog box definition. Before you insert a control without a label into a dialog box, you should insert a text control if you want to create an association between the unlabelled control and the text control. To add a shortcut key to a control that doesn't have a label:

1. Select the text control that immediately precedes the unlabelled control.

   Physical location in the dialog box is not important; it is the location of the Corel SCRIPT statement in the script that is important.

2. Click Edit, Attributes.
3. In the Label text box place an ampersand (&) in front of the letter you want to use as a shortcut key.

Notes

- A shortcut key is an alternative to using TAB to move within a dialog and change focus. To use a shortcut, press ALT+***!*** where ***!*** is the defined shortcut key.
- Shortcut keys in a dialog box should be unique.

{button ,AL(`a_focus;;;;;',0,"Defaultoverview",)} **Related Topics**

To change dialog tab order

Tab order in a dialog box is based on the order in which the controls are listed in the Corel SCRIPT script. The underlying Corel SCRIPT statements for controls are stored in the order in which they were inserted into a dialog box.

To change the tab order of a control in a dialog box

- Enter a relative position value in the Tab Order window of the dialog window's Properties Bar.

Notes

- If the Properties Bar isn't visible, click View, Properties Bar to make it visible.
- You can reorder controls in a dialog box using the dialog window's Control List Move Up and Move Down buttons. If the Controls List isn't visible, click View, Controls List to make it visible.

{button ,AL(`a_focus;ht_view_properties;ht_view_controls;;;',0,"Defaultoverview",)} Related Topics

To open on-line Help to a selected dialog control's syntax reference

1. Click Help, What's This
2. Select a control.

{button ,AL(`cse_help;;;;;',0,"Defaultoverview",)} **Related Topics**

To view or hide a dialog window's Controls List

- Click View, Controls List.

  A check mark beside the Control List menu command indicates the Control List is displayed in the active dialog window.

Note

- You can only issue this command if a dialog window is active.

{button ,AL(`ht_view_wins;;;;;',0,"Defaultoverview",)} Related Topics

File Number

An integer (whole number) value between 1-10, inclusive. Non-integers are truncated.

CLOSE statement

CLOSE #num,...

Closes a text file opened with an OPEN statement (**OPEN...APPEND** or **OPEN...OUTPUT**).

Parameter	Description
num	Numeric expression(s) specifying a file number to close. If not specified, all open files are closed. The # sign is optional.

Note

- In your scripts, every OPEN statement should have a corresponding CLOSE statement.

Examples

```
CLOSE
```
Closes all open files.

```
CLOSE #1
```
Closes the file opened as 1.

```
CLOSE 1
```
Closes the file opened as 1.

```
CLOSE 1, 3, 5
```
Closes the files opened as 1, 3, and 5.

{button ,AL(`open_append;open_input;open_output;;;',0,"Defaultoverview",)} Related Topics

EOF function

EOF (#num)

Returns TRUE (-1) if the file pointer is at the end of an open text file in your computer's memory. Returns FALSE (0) if the file contains data beyond the pointer. The statement is often used to determine whether to continue processing a file.

Parameter	Description
#**num**	Numeric expression specifying an open file number to examine. The # sign is optional.

Example

```
IF EOF(1) THEN CLOSE 1
```

If the pointer is at the end of the file, then close the file.

{button ,AL(`lof;seek;open_append;print;write;',0,"Defaultoverview",)} Related Topics

INPUT function

INPUT(bytes, #num)

Starting at the pointer, reads a number of bytes (characters) from a text file.

Parameter	Description
bytes	Numeric expression specifying the number of bytes to be read.
#**num**	Numeric expression specifying an open file number to examine. The # sign is optional.

Example

```
myExtract$ = INPUT(50, #1)
```

Reads 50 characters from file 1 and assigns them to the variable **myExtract$**.

{button ,AL(`INPUT;LINE_INPUT;SEEK;FILEPOS;EOF;LOF;',0,"Defaultoverview",`main')} Related Topics

INPUT # statement

INPUT #num, var1, var2, ...

Reads from a file to a list of variables. Values in the file are separated by commas. Character strings that include commas must be enclosed in quotation marks. The quotation marks do not appear in the variable.

Parameter	Description
#**num**	Numeric expression specifying an open file number to examine.
var1, var2, ...	The variables that receive the fields as they are read from the file.

Notes

- The number sign (#) is required in the syntax.
- Numeric data with decimals can be input only if the period (**.**) is used as the decimal separator.
- If you're converting dates, they must be in the standard International format (yy/MM/dd hh:mm:ss).
- Booleans can be input only as TRUE or FALSE.

Example

```
INPUT #1, title$, number%
```

Reads a string and a numeric variable from file 1. Assigns the values to the string variable **title$** and the integer variable **number%**. The contents of file 1 could be in either of two formats:

```
"Ottawa", 50
"Huckleberry Finn", 101
"Hamlet", 220
```

or

```
"Ottawa", 50, "Huckleberry Finn", 101, "Hamlet", 220
```

{button ,AL(`input_dollar;open_input;line_input;write;;',0,"Defaultoverview",)} **Related Topics**

LINE INPUT statement

LINE INPUT #num, string

Reads the next line from an open text file in your computer's memory into a string.

Parameter	Description
#num	Numeric expression specifying an open file number to examine.
string$	String expression specifying the string variable to hold the line from the file.

Note

- The number sign (#) is required in the syntax.

Example

```
LINE INPUT #1, MyString$
```

Reads the next line from file 1 and places the string in the variable **MyString$**.

{button ,AL(`input_dollar;input;open_input;line_input;;;',0,"Defaultoverview",)} Related Topics

LOF function

LOF(#num)

LOF (Length of File) returns the number of bytes in an open text file.

Parameter	Description
#**num**	Numeric expression specifying an open file number to examine. The # sign is optional.

Example

```
bytes% = LOF(1)
```

Sets **bytes** to the number of bytes in file 1.

{button ,AL(`eof;seek;open_append;print;write;',0,"Defaultoverview",)} **Related Topics**

OPEN...APPEND statement

OPEN fileName FOR APPEND AS #num

Opens a text file to add or append sequential data to the end of the file. Data is appended using the **PRINT** or **WRITE** statement. The FOR clause is required.

Parameter	Description
fileName	String expression specifying a filename to open. If the file doesn't exist, it is created and then opened. Drive and folder information is optional.
#**num**	Numeric expression specifying a file number to associate with the open file. The # sign is optional.

Note

- You cannot use the **COPY** or **RENAME** command on an open file.

Example

```
OPEN "oldfile" FOR APPEND AS 3
```

Opens the file OLDFILE as file number 3 to append sequential data.

{button ,AL(`input_dollar;open_append;open_input;open_output;ACCESS;CLOSE;EOF;FILEMODE;FILEPOS;freefile;input;line_input;lof;print;seek;write',0,"Defaultoverview",`main')} Related Topics

OPEN...INPUT statement

OPEN fileName FOR INPUT AS #num

Opens a text file so that data can be read from it using the **INPUT** or **LINE INPUT** statement.

Parameter	Description
fileName	String expression specifying a filename to open. An error occurs if the file doesn't exist. Drive and folder information is optional.
#**num**	Numeric expression specifying a file number to associate with the open file. The # sign is optional.

Note

- You cannot use the **COPY** or **RENAME** command on an open file.

Example

```
OPEN FileString$ FOR INPUT AS 2
```

Opens the file specified by **FileString$** for input.

{button ,AL(`input_dollar;open_append;open_input;open_output;ACCESS;CLOSE;EOF;FILEMODE;FILEPOS;freefile;input;line_input;lof;print;seek;write',0,"Defaultoverview",`main')} Related Topics

OPEN...OUTPUT statement

OPEN fileName$ FOR OUTPUT AS #num%

Opens a text file so that data can be written into it using the **PRINT** or **WRITE** statement. The FOR clause is required.

Parameter	Description
fileName	String expression specifying a filename to open. If the file doesn't exist, it is created and then opened. Drive and folder information is optional.
#**num**	Numeric expression specifying a file number to associate with the open file. The # sign is optional.

Note

- You cannot use the **COPY** or **RENAME** command on an open file.

Example

```
OPEN "c:\temp\workfile.tmp" FOR OUTPUT AS 1
```

Opens the file C:\Temp\WORKFILE.TMP for output.

{button ,AL(`input_dollar;open_append;open_input;open_output;ACCESS;CLOSE;EOF;FILEMODE;FILEPOS;freefile;input;line_input;lof;print;seek;write',0,"Defaultoverview",`main')} Related Topics

PRINT statement

PRINT #num, expression1 { , | ; } expression2 { , | ; } ...

Prints expression(s) to an open text file (**OPEN...APPEND** or **OPEN...OUTPUT**) in your computer's memory.

Parameter	Description
#num	Numeric expression specifying a file number to print to.
expression1, expression2,...	The numeric or string expressions to print to the specified file.
{ , | ; }	Lets you specify how to separate the expressions being printed to the specified file. The comma places a tab between expressions. The semi-colon does not use a space, or a comma, between printed expressions.

Notes

- Ending a **PRINT** statement without a comma or a semicolon inserts a line return at the end of the printed expression.
- Strings are not enclosed in quotation marks when printed.
- Numeric expressions have either a leading space or a negative sign.
- The number sign (#) is required in the syntax.
- Dates and time are printed using the system's regional settings. The decimal character is also dependent on the system's regional settings. In Windows 95, click Start, Setting, Control Panel, Regional settings to view and change your system settings.
- Booleans are printed as TRUE or FALSE.

Example

```
PRINT #1, "A",          'inserts a tab after A
PRINT #1, "B"           'inserts a return after B
PRINT #1, "C";          'no spacing after C
PRINT #1, "D"
```

The above example prints the following to file #1 (there is a tab between A and B):

```
A   B
CD
```

In the following example, a blank line is printed, and then the string **HEADING:** is followed by the value of **string1$**, a comma, a space, and the value in **my_int%**.

```
PRINT #1,
PRINT #1, "HEADING: "; string1$; ", "; my_int%
```

{button ,AL(`seek;print;write;open_append;open_output;input;;',0,"Defaultoverview",)} Related Topics

SEEK statement

SEEK #num, position

To prepare for subsequent input, moves the file pointer to a specific byte position in an open text file in your computer's memory. Characters, spaces, tabs, and line breaks, are counted as characters.

Parameter	Description
num	Numeric expression specifying a file number to examine. The # sign is optional.
position	Numeric expression specifying the byte position to SEEK (1 is 1st byte).

Note

- The seek always starts from the first position in the file.
- The seek range is limited from 1 to length of the file (**LOF**) + 1.

Example

```
SEEK 1, 100
```

Moves the file pointer to the 100th byte in file 1.

{button ,AL(`OPEN_APPEND;OPEN_INPUT;OPEN_OUTPUT;lof;;',0,"Defaultoverview",)} Related Topics

WRITE statement

WRITE #num, expression1 { , | ; } expression2 { , | ; } ...

Writes expressions to an open text file (**OPEN...APPEND** or **OPEN...OUTPUT**) in your computer's memory.

Parameter	Description
#num%	Numeric expression specifying a file number to write to.
expression1, expression2,...	The numeric or string expressions to print to the specified file.
{ , | ; }	Lets you specify a separator between expressions. Both the comma and the semicolon insert a comma as a separator.

Notes

- Ending a complete **WRITE** statement without a comma or a semicolon inserts a line return at the end of the written expression.
- Strings are enclosed in quotation marks when printed.
- Numeric expressions have either a leading space or a negative sign.
- The number sign (#) is required in the syntax.
- Numeric data is written using the period (**.**) as a decimal separator.
- If you're converting dates, they must be in the standard International format (yy/MM/dd hh:mm:ss).
- Booleans are written as TRUE or FALSE.

Example

```
WRITE #1, "A ",          'inserts a space and comma after A
WRITE #1, "B"            'inserts a return after B
WRITE #1, "C";           'inserts a comma after C
WRITE #1, "D"            'inserts a return after D
WRITE #1,                'prints a blank line
```

The above example prints the following to file #1:

```
"A ","B"
"C","D"
```

{button ,AL(`seek;print;write;open_append;open_output;input;;',0,"Defaultoverview",)} Related Topics

ABS function

ABS(x)

Returns the absolute value of a number. Absolute value is the positive value of a number.

Parameter	Description
x	Any numeric expression.

Examples

```
x% = ABS(-3)
y = ABS(3)
```

Both the above examples return 3 to **x** and **y**.

{button ,AL(`abs;sgn;fix;int;;',0,"Defaultoverview",)} Related Topics

ACOS function

ACOS(x)

Returns the inverse cosine (arc cosine) of a given value. The result is an angle measured in radians between 0 and π (where π is approximately 3.14152).

Parameter	Description
x	A numeric expressionbetween -1 and 1.

Note

- To convert the result from radians to degrees, use the **ANGLECONVERT** function or multiply the result by 180/3.14152.

Examples

```
v = ACOS(-0.75)
w = ACOS(0.75)
x = ACOS(0)
y = ACOS(1)
```

In the above example, **v** is equal to 2.418858406, **w** is equal to 0.722734248, **x** is equal to 1.570796327, and **y** is equal to 0.

{button ,AL(`Math_PASTE;ACOS;ASIN;ATAN;;',0,"Defaultoverview",)} Related Topics

ASIN function

ASIN(x)

Returns the inverse sine (arc sine) of a given value. The result is an angle in radians bounded by -p /2 and p /2.

Parameter	Description
x	A numeric expressionbetween -1 and 1.

Note

- To convert the result from radians to degrees, use the **ANGLECONVERT** function or multiply the result by 180/3.14152.

Examples

```
w = ASIN(-0.75)
x = ASIN(0.75)
y = ASIN(0)
z = ASIN(1)
```

In the above example, **w** is equal to -0.8480621, **x** is equal to 0.8480621, **y** is equal to 0, and **z** is equal to 1.570796.

{button ,AL(`Math_PASTE;ACOS;ASIN;ATAN;;',0,"Defaultoverview",)} Related Topics

ATAN function

ATAN(x)

Returns the inverse tangent (arc tangent) of a given value. The result is an angle bounded by -π /2 (-90 degrees) and π /2 (90 degrees) measured in radians.

Parameter	Description
x	A numeric expression.

Note

- To convert the result from radians to degrees, use the **ANGLECONVERT** function or multiply the result by 180/3.14152.

Examples

```
w = ATAN(-0.75)
x = ATAN(0.75)
y = ATAN(0)
z = ATAN(1)
```

In the above example, **w** is equal to -0.6435011, **x** is equal to 0.6435011, **y** is equal to 0, and **z** is equal to 0.7853982.

{button ,AL(`Math_PASTE;ACOS;ASIN;ATAN;;',0,"Defaultoverview",)} Related Topics

COS function

COS(x)

Returns the cosine of an angle measured in radians.

Parameter	Description
x	Any numeric expression. Lets you specify the angle measured in radians.

Notes

- The result of COS is between -1 and 1.
- To convert degrees to radians, multiply degrees by 3.14159/180 (π is approximately equal to 3.14159) or use the **ANGLECONVERT** function.

Examples

```
degreeMeasure% = 45
MyResult = COS(3.14159/180*degreeMeasure%)
```

The above example returns the COS of 45 degrees as expressed in radians. The variable **MyResult** equals 0.707106781.

{button ,AL(`Math_PASTE;CS_MATH_FNS;;;;',0,"Defaultoverview",)} Related Topics

DEC function

DEC(x)

Returns the conversion of a hexadecimal value into decimal notation (as a long data type).

Parameter	Description
x	A string expression representing a hexadecimal number.

Notes

- Decimal notation is a numerical system based on groups of ten units.
- The highest value you can convert is 7FFFFFFF.
- The **HEX** function performs the opposite conversion, from decimal to hexadecimal.

Examples

```
x& = DEC("A27")
```

In the above example, **x** equals 2599.

{button ,AL(`hex;;;;;',0,"Defaultoverview",)} Related Topics

EXP function

EXP(x)

Raises *e* to a given exponent where *e* is the base of the natural logarithm which equals 2.718281828.

Parameter	Description
x	Any numeric expression.

Note

- EXP is the inverse of the natural logarithm (**LN**).

Example

```
x = EXP(7.89)
```

In the above example, **x** equals 2670.444.

{button ,AL(`log;ln;exp;;;',0,"Defaultoverview",)} Related Topics

FIX function

FIX(x)

Removes an argument's decimal or fraction and rounds towards 0. An integer is returned.

Parameter	Description
x	A numeric expression.

Note

- Both **INT** and FIX return the integer portion of a given number. However, INT returns the greatest integer less than or equal to the number, while FIX returns the integer portion given, without any decimal points represented. As a result, -5.26 becomes -5 under the FIX function, and -6 under the INT function.

Examples

```
vv = FIX(12.65)
yy = FIX(-47.29)
```

In the above example, **vv** equals 12 and **yy** equals -47.

```
v = INT(12.65)
y = INT(-47.29)
```

In the above example, **v** equals 12 and **y** equals -48.

{button ,AL(`abs;sgn;fix;int;;',0,"Defaultoverview",)} Related Topics

HEX function

HEX(x)

Converts a number to its corresponding hexadecimal string value.

Parameter	Description
x	A numeric expressionspecifying a decimal value to convert.

Notes

- Decimal numbers are rounded to the nearest whole number before being converted.
- Hexadecimal notation is a numerical system based on groups of sixteen units. Hexadecimal numbers can be expressed in Corel SCRIPT by using the prefix **&h**. For example, &h10 or &h1ABC.
- The **DEC** function performs the opposite conversion, from hexadecimal to decimal.

Example

```
x$ = HEX(27)
x$ = HEX(27.25)
```

In the above example **x** and **y** are both passed the value "1B".

{button ,AL(`dec;;;;;',0,"Defaultoverview",)} Related Topics

INT function

INT(x)

Removes an argument's decimal or fraction and rounds down to the nearest integer. An integer is returned.

Parameter	Description
x	A numeric expression.

Note

- Both INT and **FIX** return the integer portion of a given number. However, INT returns the greatest integer less than or equal to the number, while FIX returns the integer portion given, without any decimal points represented. As a result, -5.26 becomes -5 under the FIX function, and -6 under the INT function.

Examples

```
v = INT(12.65)
y = INT(-47.29)
```

In the above example, **v** equals 12 and **y** equals -48.

```
vv = FIX(12.65)
yy = FIX(-47.29)
```

In the above example, **vv** equals 12 and **yy** equals -47.

{button ,AL(`abs;sgn;fix;int;;',0,"Defaultoverview",)} Related Topics

LN function

LN(x)

Returns the natural logarithm (base *e*) of a number.

Parameter	Description
x	Any positive numeric expression.

Notes

- LN is the inverse of the **EXP** function; therefore LN(EXP(**x**)) equals **x**.
- Natural logarithms are based on the constant *e* which is equal to 2.718281828.

Examples

```
w = LN(1.2)
x = LN(30)
y = LN(1)
z = LN(EXP(30))
```

In the above example, **w** is equal to 0.1823215568, **x** is equal to 3.401197382, **y** is equal to 0, and **z** is equal to 30.

{button ,AL(`log;ln;exp;;;',0,"Defaultoverview",)} Related Topics

LOG function

LOG(x)

Returns the base-10 logarithm of a number.

Parameter	Description
x	Any positive numeric expression.

Note

- The LOG function uses a base of 10. If another base is needed, use LOG(x)/LOG(b) formula where **b** is the base.

Examples

```
x = LOG(25)
y = LOG(5)
```

In the above example, **x** equals 1.397940 and **y** equals 0.6989700.

{button ,AL(`log;ln;exp;;;',0,"Defaultoverview",)} **Related Topics**

RANDOMIZE function

RANDOMIZE x

Sets the random number generator seed to the integer portion of the argument value.

Parameter	Description
x	Any numeric expression. This is an optional parameter. **Randomize** uses the argument as a seed to start a new sequence of random numbers. Using RANDOMIZE without an argument initializes the seed to the system timer. If RANDOMIZE isn't used before calling **RND**, the same sequence of numbers will result every time the random number generator is used.

Examples

```
RANDOMIZE
RANDOMIZE 24
```

In the above example, the first line initializes the random number generator seed to the system timer. The second line uses 24 as the first seed in the random number generator's sequence.

The following example returns 5 random integers between 1 and 10 to the variable **a_random**. Each random value is also displayed in a message box.

```
RANDOMIZE
FOR x = 1 TO 5
   lower=1
   upper=10
   a_random = INT((upper - lower +1)*RND()+lower)
   MESSAGE a_random
NEXT x
```

{button ,AL(`randomize;rnd;;;;',0,"Defaultoverview",)} Related Topics

RND function

RND(x**)**

Returns a random number.

Parameter	Description
x	A numeric expression that determines the bounds of the random number. See the table below.

x's value	Random Number Bounds
Greater than 0	A number between 0 (lower bound) and x (upper bound).
Less than 0	A number between x (lower bound) and 0 (upper bound).
Omitted	A number between 0 and 1.

Note

- If **RANDOMIZE** isn't used before calling **RND** for the first time, the same sequence of numbers will result every time the script is run.

Examples

```
x = RND()
y = RND(-7)
z = RND(24)
```

In the above example, **x** returns a random number between 0 and 1, **y** returns a random number between -7 and 0, and **z** returns a random number between 0 and 24.

To create a random integer in a specified range, use the following formula:

```
INT((upper - lower + 1)*RND()+ lower)
```

where **upper** is the highest number in the specified range and **lower** is the lowest number in the specified range. The following example returns a random integer between 1 and 10 to the variable **a_random**.

```
lower = 1
upper = 10
a_random = INT((upper - lower +1)*RND()+lower)
```

{button ,AL(`randomize;rnd;;;;',0,"Defaultoverview",)} **Related Topics**

SGN function

SGN(x)

Determines the sign (+ or -) of a number. Returns -1 if the number is negative, 1 if the number is positive, and 0 if the number is 0.

Parameter	Description
x	Any numeric expression.

Examples

```
x% = SGN(5)
y = SGN(0)
z = SGN(-0.021)
```

In the above example, **x** is equal to 1, **y** is equal to 0, and **z** is equal to -1.

{button ,AL(`abs;sgn;fix;int;;',0,"Defaultoverview",)} Related Topics

SIN function

SIN(x)

Returns the sine of an angle measured in radians.

Parameter	Description
x	Any numeric expression. Lets you specify the angle measured in radians.

Note

- The result of SIN is between -1 to 1, inclusive.
- To convert degrees to radians, multiply degrees by 3.14159/180 (π is approximately equal to 3.14159) or use the **ANGLECONVERT** function.

Examples

```
degreeMeasure% = 45
MyResult = SIN(3.14159/180*degreeMeasure%)
```

The above example returns the SIN of 45 degrees. The variable **MyResult** equals 0.70710678.

{button ,AL(`Math_PASTE;CS_MATH_FNS;;;;',0,"Defaultoverview",)} Related Topics

SQR function

SQR(x)

Returns the positive square root of a number.

Parameter	Description
x	Any non-negative numeric expression.

Examples

```
w = SQR(4)
x = SQR(0.175)
```

In this example, **w** is equal to 2 and **x** is equal to 0.4183300133.

{button ,AL(`abs;sgn;fix;int;;',0,"Defaultoverview",)} Related Topics

TAN function

TAN(x)

Returns the tangent of an angle measured in radians.

Parameter	Description
x	Any numeric expression. Lets you specify the angle measured in radians.

Note

- To convert degrees to radians, multiply degrees by 3.14159/180 (*p* • is approximately equal to 3.14159) or use the **ANGLECONVERT** function.

Example

```
radianMeasure = 0.5
MyResult = TAN(radianMeasure)
```

The above example returns the TAN of 0.5 radians. The variable **MyResult** equals 0.546302.

{button ,AL(`Math_PASTE;CS_MATH_FNS;;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT and OLE automation

Any Corel application that supports Corel SCRIPT provides a programmable OLE automation **object**. The object is used by OLE automation controllers to send Corel SCRIPT application commands to their respective Corel application. For example, Corel SCRIPT DRAW application commands are sent to CorelDRAW.

You can use OLE automation controllers such as Microsoft Visual Basic, Microsoft Visual Basic for Applications, and Microsoft Visual C++ (with the Microsoft Foundations Classes) to send commands to applications that support OLE automation objects such as CorelDRAW, and to develop applications using Corel application components.

The *Corel application commands and functions* in this online Help file provide a reference of all available commands and functions in your Corel application. The commands and functions are a part of **automation-enabled objects**. The online Help provides only overview procedural and reference information about programming with OLE automation. For more information about OLE automation, see the following reference sources:

- Microsoft Visual Basic Programmer's Guide
- Microsoft Visual Basic for ApplicationsUser's Guide
- Microsoft Windows Developer's Kit
- Microsoft Office Developer's Kit

Corel SCRIPT Editor

The Corel SCRIPT Editor is an OLE automation controller; that is, you can use it to access objects in other applications that have OLE automation objects. Ordinarily, Corel SCRIPT Editor is used to access objects in Corel applications that support OLE automation. However, you can also access objects in non-Corel applications. For example, you can call Microsoft Word 6.0 or Microsoft Excel 5.0 using the external names "Word.Basic" or "Excel.sheet.5", respectively.

To access objects in applications that support OLE automation from the Corel SCRIPT Editor, use the **WITHOBJECT** statement.

Notes

- For a list of Corel applications that support OLE automation, and their application object names, click  .
- The advanced Corel SCRIPT programming feature described in this section is intended for experienced Windows programmers, and not for beginner script writers.

{button ,AL(`ole_cs;;;;;',0,"Defaultoverview",)} Related Topics

Using Corel applications with OLE automation controllers

To access a Corel application with an OLE automation controller, an object variable must be first defined for the Corel application. Each Corel application that supports OLE automation has one object that can be accessed by a controller. For a list Corel applications that support OLE automation and their respective automation object names, click  .

The following section provides an example of using Microsoft Visual Basic to access Corel PHOTO-PAINT. Other OLE automation controllers may access OLE automation objects with different instructions.

Example

From Microsoft Visual Basic, the first step is to declare an object variable type. For example,

```
DIM paint AS OBJECT
```

The next step is to assign the application object to the object variable previously declared. In this case, the object variable is **paint,** and the Visual Basic **CreateObject** function with the PHOTO-PAINT object name is used to assign the application object. For example,

```
SET paint = CREATEOBJECT ("CorelPhotoPaint.Automation.6")
```

Corel PHOTO-PAINT commands can now be accessed by Visual Basic.

Starting applications

Before an automation controller (for example, Visual Basic) can access a Corel application that supports OLE automation objects, the application must be running. If it is not running, the controller attempts to start it. The application location and path information is stored in the Windows Registry.

You can use the Microsoft Visual Basic **Set Nothing** command to discontinue an association to a previously declared object.

One object in Corel OLE automation applications

Corel applications that support OLE automation have one object only. (For a list Corel applications that support OLE automation and their respective Corel application object names, click  .)

{button ,AL(`ole_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

Using Corel SCRIPT application commands: an example

Once an OLE automation controller has assigned a Corel application object to a variable and made it available, it can use Corel SCRIPT statements, functions, and commands the same way a Corel SCRIPT script uses them.

However, the OLE automation controller can only use Corel application statements. For example, it can use the commands **EditCut** from Corel PHOTO-PAINT and **FileOpen** from CorelDRAW, but it cannot use Corel SCRIPT programming statements and functions such as **FOR...NEXT** or **MESSAGE** or Corel dialog box definition statements. See the Corel SCRIPT programming language for more details about application commands, programming statements, and functions.

Example

The following example, which uses Microsoft Visual Basic as an automation controller and Corel PHOTO-PAINT as the OLE automation application receiving instructions, creates a Windows bitmap called SQUARE.BMP. The commands in bold are Visual Basic statements; the others are Corel SCRIPT application commands.

```
DIM paint AS OBJECT   'declare on object variable
SET paint = CREATEOBJECT ("CorelPhotoPaint.Automation.6") 'assigning the application object
    .FileNew 216, 288, 4, 72, 72, 0, 0, 1, 12643084, 12, 87753640, 11203980, 255, 255, 255, 0
    .SetPaintColor 5, 51, 255, 0, 0
    .PenSettings 20, 20, 0, 0, 0, 0, 0
    .ShapeSettings 50, 0, 20, -1, 0
    .Rectangle 52, 218, 196, 65
    .FileSave "C:\COREL60\PROGRAMS\square.BMP", 769, 0
SET paint = NOTHING   'discontinues the association to the declared object
```

{button ,AL(`ole_cs;;;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT Editor: OLE automation controller

Not only can you use the Corel SCRIPT Editor to edit and debug Corel SCRIPT scripts, but you can use it as an OLE automation controller. Any time you run a script from the Editor that contains Corel SCRIPT application statements, you are using the Editor's OLE automation capabilities to access Corel application commands.

You can also use the Editor to create scripts that can call non-Corel applications . For example, you can call Microsoft Word 6 or Microsoft Excel 5 using the OLE Automation identifiers "Word.Basic" or "Excel.sheet.5" respectively, with the **WITHOBJECT** statement.

Example

The following Corel SCRIPT script example creates a new Microsoft Word 6 document called NAMEDATE.DOC that contains the user's name and the current date:

```
WITHOBJECT "Word.Basic"
    .FileNew .Template = "Normal", .NewTemplate = 0
    .InsertField .Field = "USERNAME  \* MERGEFORMAT"
    .Insert Chr$(9)
    .InsertDateTime .DateTimePic = "dddd, MMMM dd, yyyy", .InsertAsField = 1
    .FileSaveAs .Name = "NAMEDATE.DOC", .Format = 0, .LockAnnot = 0, .Password = "", .AddToMru = 1, .WritePassword = "", .RecommendReadOnly = 0, .EmbedFonts = 0, .NativePictureFormat = 0, .FormsData = 0
END WITHOBJECT
```

The Word Basic commands, or any commands from an OLE automation application receiving instructions, must be preceded by a period. In the above example, the Word Basic **FileNew** command became the **.FileNew** command.

Corel WordPerfect Suite applications (Corel WordPerfect, Corel Quattro Pro, and Corel Presentations) do not support Corel SCRIPT. All three applications, however, do support OLE Automation. You could use the Corel SCRIPT Editor to create a script that sends instructions to any of these applications using the Corel application's macro or Perfect Script commands. The following example performs the same operation with Corel WordPerfect that the above Microsoft Word 6 example performed:

```
WITHOBJECT "WordPerfect.PerfectScript"
    .FileNew
    .CommentInsertUserName
    .Tab
    .DateText
    .FileSave "NAMEDATE.DOC"
END WITHOBJECT
```

Click ➪ for a list of Corel applications that support OLE Automation.

Notes

- You can also execute the script example shown above, or any other script that calls the automation controller from any Corel application that supports Corel SCRIPT. See To run a Corel SCRIPT script from a Corel application for more information.
- Before the Corel SCRIPT OLE automation controller can access an OLE automation application (for example, Corel WordPerfect or Microsoft Word), the application that is being accessed must be running. If it is not running, the Corel SCRIPT Editor controller attempts to start it. The application location and path information is stored in the Windows Registry.
- After the **END WITHOBJECT** command is executed, the Corel SCRIPT OLE automation controller can no longer access the object declared in the **WITHOBJECT** command until another **WITHOBJECT** command is issued.

{button ,AL(`ole_cs;corel_script_editor;;;;',0,"Defaultoverview",)} Related Topics

Arithmetic Operators

Use arithmetic operators to perform mathematical operations on two numeric expressions. Arithmetic operators are placed between two numeric expressions.

Operator	Description
+ (Addition)	Sums two numeric expressions. The + operator is also a concatenation and unary operator. Click ⇨ for an example.
- (Subtraction)	Subtracts one numeric expression from another. The - operator is also a unary operator. Click ⇨ for an example.
* (Multiplication)	Multiplies two numeric expressions. Click ⇨ for an example.
/ (Division)	Divides two numeric expressions returning a fractional result. Click ⇨ for an example.
^ (Exponentiation)	Raises a numeric expressions to the power of another numeric expression. Click ⇨ for an example.
MOD (Modulus)	Returns a whole number remainder of division between two numeric expressions. Non-integer expressions are rounded before division. Click ⇨ for an example.
\ (Integer division)	Divides two numeric operands and returns a whole number. Non-integer expressions are rounded before division to whole numbers. Click ⇨ for an example.

Note

- All arithmetic operators have a left-associativity, except the exponentiation operator (^). Left-associativity means the expression with the operator is evaluated from left to right. For example, **4 * 10 / 2** is calculated as 4 times 10 equals 40 divided by 2 equals 20. For the exponentiation operator which has a right-associativity, the expression is calculated from right to left. For example, the expression **4^3^2** is calculated as 3 to the power of 2 (which equals 9), and then 4 to the power of 9, which equals 262,144.

{button ,AL(`all_operators;;;;;',0,"Defaultoverview",)} Related Topics

+ (Addition) Example

```
i = a + b
```

Assigns the sum of the values of variables **a** and **b** to the variable **i**.

- (Subtraction) Example

```
k = k - 2
```

Decreases the variable **k** by 2.

* (Multiplication) Example

```
k = (c + d) * 5
```

Adds the values of variables **c** and **d** together, multiplies the sum by 5, and assigns the result to the variable **k**. Note that the parentheses force the calculation of the addition before the multiplication.

* (Division) Example

```
m = e / f
```

Assigns the result of the value of variable **e** divided by the value of **f** to the variable **m**.

^ (Exponentiation) Example

```
n = g ^ 2
```

Assigns the result of the value of **g** raised to the second power to the variable **n**. The expression **g ^ 2** is the same as **g * g**.

MOD (Modulus) Example

```
n = 14 MOD 3
n = 13.9 MOD 3.6
```

Both of the examples assign the value of 2 to **n**. In the second example, the operands are rounded to 14 and 3.

\ (Integer division) Example

```
n = 15 \ 4
n = 15.88 \ 4
n = 15 \ 4.123
n = 15.88 \ 4.123
```

The first and third examples assign the value of 3 to **n**. The second and fourth examples assign the value of 4 to **n**. In the second, third, and fourth examples, the operands are rounded to 16 and 4, respectively.

Assignment Operator

Use the assignment operator (=) to assign the value of a right operand expression to a left operand variable. The = operator is also a relational operator.

In the following example, **fox** equals the number 1996 and **car** is equal to the string "Ford"

```
fox = 1987 + 9
car = "Ford"
```

Note

- The **LET** statement can also be used with the assignment operator.

{button ,AL(`all_operators;LET;;;;',0,"Defaultoverview",)} Related Topics

Bitwise Operators

Use bitwise operators to compare identically positioned bits in expressions holding integers. Non-integer expressions are rounded.

Operator	Description
NOT (logical complement)	Unlike the three other bitwise operators, NOT is unary, it only operates on one operand (see Unary Operators for more information). NOT returns the opposite of the corresponding bits of the operand, which is sometimes called "one's-complement." Click ➪ for an example. The NOT operator is also a logical operator.
AND (conjunction)	Compares the individual bits in the expression to the left of the operator with the individual bits in the expression to the right of the operator. If the identically positioned bits are both 1, 1 is returned. If not, 0 is returned. Click ➪ for an example. The AND operator is also a logical operator.
OR (disjunction)	Compares the individual bits in the expression to the left of the operator with the individual bits in the expression to the right of the operator. If either of the identically positioned bits are 1, 1 is returned. If not, 0 is returned. Click ➪ for an example. The OR operator is also a logical operator.
XOR (exclusive or)	Compares the individual bits in the expression to the left of the operator with the individual bits in the expression to the right of the operator. If the identically positioned bits are both 1 or both 0, 0 is returned. If not, 1 is returned. Click ➪ for an example. The XOR operator is also a logical operator.
EQV (equivalence)	Compares the individual bits in the expression to the left of the operator with the individual bits in the expression to the right of the operator. If the identically positioned bits are both 1 or both 0, 1 is returned. If not, 0 is returned. Click ➪ for an example. The EQV operator is also a logical operator.
IMP (implication)	Compares the individual bits in the expression to the left of the operator with the individual bits in the expression to the right of the operator. If the identically positioned bits are 1 and 0, respectively, 0 is returned. If not, 1 is returned. Click ➪ for an example. The IMP operator is also a logical operator.
<< (Shift Left)	Shifts bits a specified number of bit positions to the left, inserting 0 bits at the right end of the binary numeral. Click ➪ for an example.
>> (Shift Right)	Shifts bits a specified number of bit positions to the right, inserting 0 bits at the left end of the binary numeral. Click ➪ for an example.

Notes

- The operators are listed in their order of precedence.
- Although bitwise and logical operators are listed separately, they are the same operators, but function differently depending on the operand.

{button ,AL(`all_operators;;;;;',0,"Defaultoverview",)} Related Topics

Bitwise NOT Example

```
n% = NOT(a%)
```

Toggles the bit values in **a%** to set **n%**. By toggling, bits equal to 1 are set to 0, and bits equal to 0 are set to 1. For example, if **a%** = 9, **n%** = -10 as follows:

```
0 0 0 0 1 0 0 1 = 9

1 1 1 1 0 1 1 0 = -10
```

The above example only uses eight bits to demonstrate the bit operation, but Corel SCRIPT integers are 16 bits long. Bit positions are counted right-to-left and the first position is called the 0 position.

Bitwise AND Example

```
n% = a% AND b%
```

Compares the bits of the variables **a%** and **b%** and sets the corresponding bits of **n%** to 1 if the same bit is 1 in both **a%** and **b%**. For example, if **a%** = 9 and **b%** = 12, then **n%** = 8 as follows:

```
0 0 0 0 1 0 0 1 = 9
0 0 0 0 1 1 0 0 = 12
0 0 0 0 1 0 0 0 = 8
```

Because only the 3rd bit (bits are always counted from right to left, starting with 0) is 1 in both operands, only the resulting number has the 3rd bit set to 1.

The above example only uses eight bits to demonstrate the bit operation, but Corel SCRIPT integers are 16 bits long. Bit positions are counted right to left and the first position is called the 0 position.

Bitwise OR Example

```
n% = a% OR b%
```

Compares the bits of the variables **a%** and **b%**, and sets the corresponding bits of **n%** to 1 if the same bit is 1 in either **a%** or **b%**. For example, if **a%** = 9 and **b%** = 12, then **n%** = 13 as follows:

```
0 0 0 0 1 0 0 1 = 9
0 0 0 0 1 1 0 0 = 12
0 0 0 0 1 1 0 1 = 13
```

The 3rd, 2nd, and 0 bits (bits are always counted from right to left, starting with 0), are set to 1 because those bits have a value of 1 in either of the two operands.

The above example only uses eight bits to demonstrate the bit operation, but Corel SCRIPT integers are 16-bits long. Bit positions are counted right-to-left and the first position is called the 0 position.

Bitwise XOR Example

```
n% = a% XOR b%
```

Compares the bits of the variables **a%** and **b%** and sets the corresponding bits of **n%** to 1 if the same bit is different in **a%** and **b%**. For example, if **a%** = 9 and **b%** = 12, then **n%** = 5 as follows:

```
0 0 0 0 1 0 0 1 = 9
0 0 0 0 1 1 0 0 = 12
0 0 0 0 0 1 0 1 = 5
```

Because the 3rd bit (bits are always counted from right to left, starting with 0) is 1 in both operands, the 3rd bit in the result is set to 0. The 2nd and 0 bits are different in the two operands, so those bits are set to 1 in the result.

The above example only uses eight bits to demonstrate the bit operation, but Corel SCRIPT integers are 16 bits long. Bit positions are counted right to left and the first position is called the 0 position.

Bitwise EQV Example

```
n% = a% EQV b%
```

Compares the bits of the variables **a%** and **b%** and sets the corresponding bits of **n%** to 1 if the same bit is equal in **a%** and **b%**. For example, if **a%** = 9 and **b%** = 12, then **n%** = 10 as follows:

```
0 0 0 0 1 0 0 1 = 9
0 0 0 0 1 1 0 0 = 12
0 0 0 0 1 0 1 0 = 10
```

Because the 3rd bit (bits are always counted from right to left, starting with 0) is 1 in both operands, the 3rd bit in the result is set to 1. The 2nd and 0 bits are different in the two operands, so those bits are set to 0 in the result.

The above example only uses eight bits to demonstrate the bit operation, but Corel SCRIPT integers are 16 bits long. Bit positions are counted right to left and the first position is called the 0 position.

Bitwise IMP Example

```
n% = a% IMP b%
```

Compares the bits of the variables **a%** and **b%** and sets the corresponding bits of **n%** to 1 unless the bit in **a%** is set to 1 and the bit in **b%** is set to 0. For example, if **a%** = 9 and **b%** = 12, then **n%** = 14 as follows:

```
0 0 0 0 1 0 0 1 = 9
0 0 0 0 1 1 0 0 = 12
0 0 0 0 1 1 1 0 = 14
```

Because the 0 bit (bits are always counted from right to left, starting with 0) is 1 in the first operand, and the 0 bit is 0 in the second operand, the 0 bit in the result is set to 0. All other bits are set to 1.

The above example only uses eight bits to demonstrate the bit operation, but Corel SCRIPT integers are 16 bits long. Bit positions are counted right to left and the first position is called the 0 position.

Bitwise Shift Left (<<) Example

```
n% = 1 << a%
```

This example shifts the bits in **a%** one position left, and inserts one 0 bit at the right end of the binary numeral. For example, if **a%** = 500, **n%** = 1000

```
0 1 1 1 1 1 0 1 0 0 = 500
1 1 1 1 1 0 1 0 0 0 = 1000
```

Bitwise Shift Right (>>) Example

```
n% = 1 >> a%
```

This example shifts the bits in **a%** one position right, and inserts one 0 bit at the left end of the binary numeral. For example, if **a%** = 1000, **n%** = 500

```
1 1 1 1 1 0 1 0 0 0 = 1000
0 1 1 1 1 1 0 1 0 0 = 500
```

Integer

Integers include all positive whole numbers, their negatives, and zero, e.g., -3, 0, 5.

Concatenation Operators

Concatenation operators join strings together to make a single string.

Operator	Description
&	Links strings or string variables together.
+	Links strings or string variables together. The + operator is also a unary and arithmetic operator.

Notes

- You can only use the concatenation operators with strings.
- Use the **STR** function to return numbers as strings.

{button ,AL(`all_operators;;;;;',0,"Defaultoverview",)} Related Topics

Examples of Concatenation Operators

& operator

```
firstString$ = "Corel"
secondString$ = "DRAW"
result$ = firstString$ & " " & secondString$
```

Assigns the value of **firstString$** followed by the value of **secondString$** to the variable **result$** with a space in between. In this example, **result$** = "Corel DRAW".

+ operator

```
firstString$ = "Corel"
secondString$ = "PHOTO-PAINT"
result$ = firstString$ & " " & secondString$
```

Assigns the value of **firstString$** followed by the value of **secondString$** to the variable **result$** with a space in between. In this example, **result$** = "Corel PHOTO-PAINT".

Using the STR function

```
a%=6
firstString$ = "Corel"
secondString$ = "DRAW"
result$ = firstString$ & " " & secondString$ & STR(a%)
```

In this example, **result$** = "Corel DRAW 6".

Note

- The variables in the examples above all use type-declaration suffixes (% and $). It is not necessary to use type-declaration suffixes with these operators.

{button ,AL(`all_operators_examples;all_operators;;;;',0,"Defaultoverview",)} Related Topics

Logical Operators

Logical operators, in conjunction with relational operators, determine a relationship between two or more expressions and return a Boolean TRUE (-1) or FALSE (0) value. In Corel SCRIPT, logical operators are most often used inside of flow control structures such as DO...LOOP, IF...THEN...ELSE...ENDIF, and WHILE...WEND. The following table lists the Corel SCRIPT logical operators in their order of precedence.

Operator	Description
NOT (logical complement)	Evaluates the expression to the right of the operator. If the expression evaluates to FALSE, TRUE is returned. The NOT operator is also a bitwise and unary operator. Click ➪ for an example.
AND (conjunction)	Logically compares the value of the expression to the left of the operator with the value of the expression to the right of the operator. If both expressions evaluate to TRUE, TRUE is returned. The AND operator is also a bitwise operator. Click ➪ for an example.
OR (disjunction)	Logically compares the value of the expression to the left of the operator with the value of the expression to the right of the operator. If either of the expressions evaluate to TRUE, TRUE is returned. The OR operator is also a bitwise operator. Click ➪ for an example.
XOR (exclusive or)	Logically compares the value of the expression to the left of the operator with the value of the expression to the right of the operator. If only one of the expressions evaluates to TRUE, TRUE is returned. The XOR operator is called the exclusive operator and is also a bitwise operator. Click ➪ for an example.
EQV (equivalence)	Logically compares the value of the expression to the left of the operator with the value of the expression to the right of the operator. If both of the expressions evaluate to TRUE or both of the expressions evaluate to FALSE, TRUE is returned. The EQV operator is called the equivalence operator and is also a bitwise operator. Click ➪ for an example.
IMP (implication)	Logically compares the value of the expression to the left of the operator with the value of the expression to the right of the operator. If the first expression evaluates as a Boolean, it implies the second expression evaluates as the same Boolean. In this case, TRUE is returned. The IMP operator is called the implication operator and is also a bitwise operator. Click ➪ for an example.

For more information about return values, see Logical Operator Return Values.

Note

- Although bitwise and logical operators are listed separately, they are the same operators, but function differently depending on the operand.

{button ,AL(`all_operators;Logical_Operator_Return_Values;;;;',0,"Defaultoverview",)} Related Topics

Logical Operator Return Values

The following table lists the conditions and corresponding Boolean return values for each Corel SCRIPT logical operator.

Conditions for NOT	Returns
Expression1 is TRUE	FALSE
Expression1 is FALSE	TRUE

Conditions for AND	Returns
Expression1 is TRUE and Expression2 is TRUE	TRUE
Expression1 is TRUE and Expression2 is FALSE	FALSE
Expression1 is FALSE and Expression2 is TRUE	FALSE
Expression1 is FALSE and Expression2 is FALSE	FALSE

Conditions for OR	Returns
Expression1 is TRUE and Expression2 is TRUE	TRUE
Expression1 is TRUE and Expression2 is FALSE	TRUE
Expression1 is FALSE and Expression2 is TRUE	TRUE
Expression1 is FALSE and Expression2 is FALSE	FALSE

Conditions for XOR	Returns
Expression1 is TRUE and Expression2 is TRUE	FALSE
Expression1 is TRUE and Expression2 is FALSE	TRUE
Expression1 is FALSE and Expression2 is TRUE	TRUE
Expression1 is FALSE and Expression2 is FALSE	FALSE

Conditions for EQV	Returns
Expression1 is TRUE and Expression2 is TRUE	TRUE
Expression1 is TRUE and Expression2 is FALSE	FALSE
Expression1 is FALSE and Expression2 is TRUE	FALSE
Expression1 is FALSE and Expression2 is FALSE	TRUE

Conditions for IMP	Returns
Expression1 is TRUE and Expression2 is TRUE	TRUE
Expression1 is TRUE and Expression2 is FALSE	FALSE
Expression1 is FALSE and Expression2 is TRUE	TRUE
Expression1 is FALSE and Expression2 is FALSE	TRUE

{button ,AL(`all_operators;;;;;',0,"Defaultoverview",)} **Related Topics**

Logical NOT Example

```
IF NOT(a% = 0) THEN BEEP
```

If the value of **a%** is not equal to 0, a beep sounds. The parentheses indicate that NOT operates on **a% = 0** and not **a%** alone.

Logical AND Example

```
IF (a% = 0) AND (b% = 0) THEN BEEP
```

If the value of **a%** is equal to 0 and the value of **b%** is equal to 0, a beep sounds. Both expressions must be true for the beep to sound.

```
IF a$ = "Corel" AND b$ = "Corel" THEN BEEP
```

If the value of **a$** is equal to "Corel" and the value of **b$** is equal to "Corel", a beep sounds. Both expressions must be true for the beep to sound.

Parentheses are placed around the relational expressions, but are not required because the **=** operator takes precedence over the OR operator. However, it is still a good practice to put parentheses around your expressions because it makes your scripts easier to read.

Logical OR Example

```
IF (a% = 0) OR (b% = 0) THEN BEEP
```

If either the value of **a%** is equal to 0 or the value of **b%** is equal to 0, a beep sounds. Only one of the expressions must be true for the beep to sound.

```
IF (a$ = "Corel") OR (b$ = "Corel") THEN BEEP
```

If either the value of **a$** is equal to "Corel" or the value of **b$** is equal to "Corel", a beep sounds. Only one of the expressions must be true for the beep to sound.

Parentheses are placed around the relational expressions, but are not required because the **=** operator takes precedence over the OR operator. However, it is still a good practice to put parentheses around your expressions because it makes your scripts easier to read.

Logical XOR Example

```
IF (a% = 0) XOR (b% = 0) THEN BEEP
```

If the value of **a%** is equal to 0 and the value of **b%** is not equal to 0, or vice versa, a beep sounds. One expression must be true and the other false for the beep to sound.

Parentheses are placed around the relational expressions, but are not required because the **=** operator takes precedence over the OR operator. However, it is still a good practice to put parentheses around your expressions because it makes your scripts easier to read.

Logical EQV Example

```
IF (a% = 0) EQV (b% = 0) THEN BEEP
```

If the value of **a%** is equal to 0 and the value of **b%** is equal to 0, a beep sounds. If both expressions are not equal to 0, a beep will also sound. Both expressions must be true or both expressions must be false for a beep to sound.

Parentheses are placed around the relational expressions, but are not required because the **=** operator takes precedence over the OR operator. However, it is still a good practice to put parentheses around your expressions because it makes your scripts easier to read.

Logical IMP Example

```
IF (a > b) IMP (b > c) THEN BEEP
```

If **a** is greater than **b** and **b** is greater than **c**, this example returns true and a beep sounds. The reverse is also true.

Relational Operators

The relational operators are used to compare two operands, numeric or string expressions of the same type, and return a TRUE (-1) or FALSE (0) value. Relational operators are most often used inside of Corel SCRIPT flow control structures such as DO...LOOP, IF...THEN...ELSE...ENDIF, and WHILE...WEND.

Operator	Description
=	Equal to. Click ➪ for an example. The = operator is also a assignment operator.
<>	Not equal to. Click ➪ for an example.
>	Greater than. Click ➪ for an example.
<	Less than. Click ➪ for an example.
>= (or =>)	Greater than or equal to. Click ➪ for an example.
<= (or =<)	Less than or equal to. Click ➪ for an example.

Note

- Because the equal sign (=) is used for both assignment and comparison, Corel SCRIPT will recognize the comparison operator only when it is within an IF, DO...LOOP, or WHILE...WEND statement.
- You can compare numeric expressions of different data types with comparison operators. Variants are treated as their data subtype in comparison operations. See the Corel SCRIPT data type summary for more information.
- Strings are always compared in their entirety, and case is significant. When comparing strings, the ANSI value of the characters in the strings are compared. See the Corel SCRIPT Character Map for more information.

{button ,AL(`all_operators;;;;;',0,"Defaultoverview",)} Related Topics

= (Equal to) Example

```
IF a% = b% THEN BEEP
```

The computer beeps if both **a%** and **b%** hold the same value. Because the equal sign (=) is used for both assignment and comparison, Corel SCRIPT will recognize the comparison operator only when it is within an IF, DO...LOOP, or WHILE...WEND statement.

<> (Not equal to) Example

```
IF str1$ <> str2$ THEN BEEP
```

The computer beeps if the values of the two string variables are not equal. Strings are always compared in their entirety, and case is significant. So if **str1$** is "Cat" and **str2$** is "cat", a beep will sound, because the strings are different.

> (Greater than) Example

```
IF a% > b% THEN BEEP
```

The computer beeps if the value of **a%** is greater than the value of **b%**.

```
IF str1$ > str2$ THEN BEEP
```

The computer beeps if the value of **str1$** is greater than the value of **str2$**. When comparing strings, the ANSI value of the characters in the strings are compared. So if **str1$** is "dog" and **str2$** is "cat", a beep will sound, because "c" comes before "d" in the ANSI character set.

< (Less than) Example

```
IF a% < b% THEN BEEP
```

The computer beeps if the value of **a%** is less than the value of **b%**.

```
IF str1$ < str2$ THEN BEEP
```

The computer beeps if the value of **str1$** is less than the value of **str2$**. When comparing strings, the ANSI value of the characters in the strings are compared. So if **str1$** is "cat" and **str2$** is "dog", a beep will sound, because "c" comes before "d" in the ANSI character set.

>= or => (Greater than or equal to) Example

```
IF a% >= b% THEN BEEP
```

The computer beeps if **a%** is greater than or equal to **b%**.

<= or =< (Less than or equal to) Example

```
IF a$ <= b$ THEN BEEP
```

The computer beeps if **a$** is less than or equal to **b$**.

Unary Operators

Unary operators perform an operation on only one operand or expression.

Operator	Description
+	Multiplies a numeric operand by +1. The + operator is also a concatenation and arithmetic operator.
-	Multiplies a numeric operand by -1. The operator is used to indicate a numeric operand is negative. The - operator is also an arithmetic operator.
NOT	Inverts the result of relational and logical expressions. The NOT operator is also a logical and bitwise operator.

{button ,AL(`all_operators;;;;;',0,"Defaultoverview",)} Related Topics

Examples of Unary Operators

+

```
x% = 12 / +3
```

-

```
y% = 12 / -3
```

NOT

```
a% = 5
b% = NOT(a% > 3)
```

Since **a%** is greater than 3, resulting in TRUE, **b%** is FALSE (or 0) because NOT inverts the result of the logical operation.

```
x = 5
z = (x < 10)
IF (NOT(z)) THEN BEEP
```

The result of expression x < 10 is assigned to variable **z**. BEEP causes the computer to beep if the inverted value of **z** equals TRUE. Because the inverted value of **z** equals FALSE, there is no beep.

Note

- The variables in the examples above all use type-declaration suffixes (% and $). It is not necessary to use type-declaration suffixes with these operators.

{button ,AL(`all_operators_examples;all_operators;;;;',0,"Defaultoverview",)} **Related Topics**

Introduction to scripts and Corel SCRIPT

A script is a computer program that executes a series of instructions with a single command. Generally, scripts are used to automate repetitive tasks, or simplify complicated actions; they can also be used to prompt for user input, display messages, and interact with other applications.

Scripts can significantly increase your productivity with Corel applications such as CorelDRAW or VENTURA, by automating repetitive tasks. For example, a script could be used to open a group of files, perform a set of editing actions, or set an application's default properties. In their simplest form, scripts replicate a Corel application's keystrokes, and toolbar, menu, and mouse commands. In a more complex form, scripts can include the commands and constructs of a programming language. For example, you could create a script that only replicates an application's commands once a series of logical requirements have been met.

A script is a text file that lists the Corel SCRIPT commands that will perform a particular task. These instructions are all part of the Corel SCRIPT programming language, which is partially based on Corel application menu commands. For example, the **FileClose** command corresponds to a menu command (click File, Close) on a Corel application's menu system.

The rest of the Corel SCRIPT programming language is based on the BASIC programming language. If you're already familiar with a version of BASIC, you'll find the Corel SCRIPT programming language easy to read and understand.

Computer programming experience isn't a prerequisite for using Corel SCRIPT to create and edit scripts. However, the more knowledge, experience, and desire you have to delve into the mechanics of your Corel application, the more you'll be able to take advantage of the power of Corel SCRIPT.

Note

- Most large computer applications have a built-in programming language of some form, but some call their programs macros instead of scripts.
- Not every Corel application supports Corel SCRIPT programming and scripts. Click for a list of Corel applications that support Corel SCRIPT.

{button ,AL(`Corel SCRIPT file types;intro_cs;corel_script_editor;corel_script_dialog_editor;;;',0,"Defaultoverview",)}
Related Topics

What is Corel SCRIPT?

Corel SCRIPT is an application included with some Corel applications to record, play, and run recordings and scripts. Click for a list of Corel applications that support Corel SCRIPT.

Corel SCRIPT starts automatically when you record or play a recording from a Corel application that supports scripts. If you run a script for an application that is not running, Corel SCRIPT automatically starts the application.

A typical user can take advantage of Corel SCRIPT to automate repetitive tasks with scripts written in the Corel SCRIPT programming language. However, Corel SCRIPT is more than just a language that you can use to create and run application script files; it's also a powerful programming language that can be used as a standalone application.

The Corel SCRIPT programming language is based on the BASIC programming language. If you're already familiar with a version of BASIC, you'll find the Corel SCRIPT programming language easy to read and understand.

Corel SCRIPT Editor

Included with Corel SCRIPT is the Corel SCRIPT Editor which you can use to create, edit, run, test, and debug a script. Though scripts are text files, and can be edited or created with almost any Windows-based text editor or word processor, you must use the Corel SCRIPT Editor if you want to test or debug a script.

Corel SCRIPT and dialog boxes

In many cases, you'll need to get information from the user before your script performs a desired action. For simple information, you can use the Corel SCRIPT function **INPUTBOX** to get a string of characters from the user returned to a running script. If you want to provide the user with options and more complex information, such as a list of choices, you can use a custom dialog box in your script.

Dialog boxes are created using the Corel SCRIPT language. Corel SCRIPT features a full set of programming statements to produce dialog boxes which incorporate sophisticated Windows options and features. You can also use Corel SCRIPT to create dynamic dialog boxes. Dynamic dialog boxes change their content depending on a user's action in a dialog box.

The Corel SCRIPT Editor includes tools to quickly create and edit dialog boxes. Working with the Editor is similar to using a drawing or painting application: dialog controls are graphic objects which can be inserted, moved, re-sized, and aligned in a dialog box. Using the Corel SCRIPT Editor takes the place of editing Corel SCRIPT dialog box statements in a script file.

Corel SCRIPT advanced features

- Corel SCRIPT is OLE Automation enabled.
- Corel SCRIPT can compile scripts as standalone executables.
- Corel SCRIPT can access functions in Dynamic link libraries (DLL), and compile standard DLLs.

Note

- Not every Corel application supports Corel SCRIPT programming and script files. Click for a list of Corel applications that support Corel SCRIPT.

{button ,AL(`Corel SCRIPT file types;intro_cs;using_dynamic_link_libraries;ole_automation;;;',0,"Defaultoverview",)}
Related Topics

Sample script

The following is an example of a script that can be used with CorelDRAW 6. The following script creates an ellipse that is filled with a fountain color. The ellipse is then duplicated three times, and the duplicates are skewed.

```
WITHOBJECT "CorelDraw.Automation.6"   'Line 1
    .CreateEllipse -250000, -500000, 250000, 500000, 0, 0, 0  'Line 2
    .ApplyFountainFill 0, -25, 17, 0, 20, 0, 3, 50  'Line 3
    .SetFountainFillcolor  0, 5, 15, 82, 155, 0  'Line 4
    .SetFountainFillcolor 20, 5, 75, 51, 225, 0  'Line 5
    .ApplyOutline 1, 0, 0, 0, 0, 0, 0, 0, 0  'Line 6
    FOR x = 1 TO 3  'Line 7
        .DuplicateObject  'Line 8
        .SkewObject -12000000, 10000000, x  'Line 9
    NEXT x  'Line 10
END WITHOBJECT  'Line 11
```

If you're an inexperienced script (or macro) writer, the above script can seem somewhat overwhelming at first, but if you break it down into its basic components (each command and statement), you may find that the script is not as complicated as it looks.

Line 1
Instructs Corel SCRIPT that the application commands which follow should be executed in CorelDRAW 6.

Line 2
Create an ellipse.

Lines 3, 4, and 5
Defines and applies a two-color fountain fill to the ellipse.

Line 6
Applies an outline to the ellipse.

Lines 7 and 10
Instructs Corel SCRIPT to repeat the commands between lines 7 and 10 three times.

Line 8
Duplicates the selected ellipse, three times.

Line 9
Skews the selected ellipse, three times.

Line 11
Instructs Corel SCRIPT to stop sending commands to CorelDRAW 6.

{button ,AL(`Introduction_Scripts_Corel_script;What_is_Corel_SCRIPT;Corel_SCRIPT_syntax;Documentation_syntax_conventions;;',0,"Defaultoverview",)} **Related Topics**

Corel SCRIPT programming language

The Corel SCRIPT programming language is made of two distinct sets of instructions:

- Corel SCRIPT application commands and functions
- Corel SCRIPT programming statements and functions

Application commands and functions

Application commands perform tasks in a specific Corel application. For example, the **FileNew** command creates a new document. Each Corel application that supports Corel SCRIPT has a distinct set of commands. Any script you create by saving a recording of your actions is made of Corel SCRIPT application commands.

The application commands are easy to understand since most are one-word equivalents of the corresponding Corel application user-interface. For example, the **EditCut** command is the complement of a menu command (click Edit, Cut) on a Corel application's menu system. Customizing your menu structure doesn't affect Corel SCRIPT command names.

Unlike the **EditCut** command, most application commands require more than just a menu name to have the command carried out. Many menu commands in Corel applications open dialog boxes with dialog controls that require user input values. Since dialog boxes are not displayed during script execution, the values required for commands that open dialog boxes must be specified with the command in the script file. These specified values are called parameters and usually correspond to dialog box options.

For example, if you wanted to open a file named **myfile.txt** in a Corel application, you might use a command similar to the following:

```
.FileOpen "myfile.txt"
```

where **"myfile.txt"** is a parameter for **FileOpen**. For more information, see Syntax for application commands and functions

An individual application command, or a block of application commands, must be enclosed by the Corel SCRIPT **WITHOBJECT** and **ENDWITHOBJECT** statements. These statements identify the Corel application that the application commands will be performed in. See the **WITHOBJECT** statement for more information.

Application functions are not recordable; they must be written into a script. Application functions ask questions about the status of a Corel application, selected items in Corel applications, or document properties. For example, a function may ask a Corel application about a document's page size. Functions cannot be recorded.

In Corel SCRIPT Help, application statements and functions are in initial caps, such as **FileOpen**, **EditCut**, **FilePrint**.

Programming statements and functions

Corel SCRIPT programming statements and functions are a common set of instructions that can be used with any Corel application that supports scripting. Programming statements and functions are derived from traditional BASIC programming language dialects and perform instructions that are not part of a Corel application. For example, Corel SCRIPT programming statements can be used to display a user-defined dialog box, include flow control constructs such as loops, create and manipulate variables, and retrieve information about your computer setup. On their own, Corel SCRIPT programming statements form a powerful programming language. In fact, a script containing only Corel SCRIPT programming statements can be executed, even if a supporting Corel application is not running.

In the Corel SCRIPT online Help, Corel SCRIPT programming statements and functions appear in uppercase, for example, **LEFT**, **IF**, and **MESSAGEBOX**. Click for a listing of all programming statements and functions.

Notes

- From the Corel SCRIPT Editor you can open Corel SCRIPT online Help to a function's, command's or statement's syntax reference, by placing the insertion point in the keyword you want help for, and pressing F1.
- Click for a list of Corel applications that support Corel SCRIPT.
- Corel SCRIPT is a command-based language that records keystrokes and mouse actions as commands. It does not record the actual keystrokes. For example, to open a file click File, Open. If you record these steps, your script will contain an application command called **FileOpen.**

{button ,AL(`intro_cs;Corel_SCRIPT_syntax;Syntax for application commands and functions;Syntax for programming statements and functions;Documentation_syntax_conventions;',0,"Defaultoverview",)} Related Topics

Creating and modifying scripts

The easiest way to create a script is to record your actions in a Corel application. You can then save the recorded actions as a script. Saving your recording as a script translates the recorded actions into script commands which are stored in a text file. When the script is executed or run, each line instructs the Corel application to perform a command turning your sequence of recorded actions into a single mouse click.

Click for a list of Corel applications that support Corel SCRIPT and recordings.

Once a recorded script is saved, it can be modified with the Corel SCRIPT Editor. For example, if you have a script that opens several documents and executes a variety of commands, you can change the documents the script opens by modify a few script commands, rather than re-recording the entire script. Using the Editor, you can also include application commands that cannot be recorded or include programming statements to customize your script. You can also use the Corel SCRIPT Editor to write scripts from scratch.

{button ,AL(`Corel_SCRIPT_Editor;;;;;',0,"Defaultoverview",)} Related Topics

Executing scripts

Corel SCRIPT scripts can be executed or run, under specific conditions, from any Corel application that supports Corel SCRIPT, or from the Corel SCRIPT Editor.

Scripts containing Corel SCRIPT application commands

Any script file that contains application commands must include the **WITHOBJECT** statement. The **WITHOBJECT** statement directs the executing script to the Corel application to call.

The following example shows how to run a **FileNew** command in Corel PHOTO-PAINT.

```
WITHOBJECT "CorelPhotoPaint.Automation.6"
    .FileNew 360, 504, 1, 72, 72, 0, 0, 1, 0, 0, 0, 19533528, 255, 255, 255, 0
END WITHOBJECT
```

You can have as many commands as you want after the **WITHOBJECT** statement, but the block of commands must end with an **END WITHOBJECT** statement. You can also have Corel SCRIPT programming statements within the **WITHOBJECT** block of commands.

You have the option of running scripts from the Corel SCRIPT Editor, or from your Corel application that supports Corel SCRIPT. Click ⇨ for a list of Corel applications that support Corel SCRIPT.

When you create or edit a script, you should first run it from the Corel SCRIPT Editor. Executing from the Editor lets you take advantage of the Editor's testing and debugging features that allow you to fine tune script syntax, or fix script syntax errors. However, once you are satisfied that your script is running properly, you should run your script from the Corel application that uses the application commands. Running your script from the application can significantly decrease the script's execution time.

You can also nest application commands for different Corel applications within a script. For example, you can create a script that copies an object from PHOTO-PAINT to CorelDRAW:

```
WITHOBJECT "CorelPhotoPaint.Automation.6"
    'Series of PHOTO-PAINT commands to copy an object
    WITHOBJECT "CorelDraw.Automation.6"
        'Series of DRAW commands to paste an object
    END WITHOBJECT
END WITHOBJECT
```

Tip

- You can also execute a script by double-clicking on its icon in the Windows Explorer or in a Windows folder.

Notes

- Scripts containing only programming statements can be run from any Corel application that supports Corel SCRIPT, or from the Corel SCRIPT Editor. Generally, a script that contains only programming statements runs faster from the Corel SCRIPT Editor.
- Unlike script files (or macro files) from other companies, Corel SCRIPT files are text only; there is no compiled binary component in the scripts. Before a script is executed, it is compiled internally into a program file.

{button ,AL(`Creating and modifying scripts_cs;Executing_script_files;Executing applications commands in the background;script_files;Script_planning_designing;',0,"Defaultoverview",)} Related Topics

Executing applications commands in the background

If you execute a script from the Corel SCRIPT Editor (or from a Corel application) containing commands for a Corel application that is not already running, Corel SCRIPT attempts to start the application. If the application is started, it is only opened in a portion of the computer memory called the *background*. Applications opened in the background are not visible on the Windows desktop.

In Windows 95, pressing CTRL+ALT+DEL opens the Close Program dialog box, which lists the 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.

Corel SCRIPT executes commands for applications only open in the background, in the same manner as for visible applications.

The following should be kept in mind when executing a script for a Corel application that is not already open:

- The first script application command after the **WITHOBJECT** statement should be a .**FileNew** or .**FileOpen** command.
- If an error occurs while your script is executing, the hidden application becomes visible.
- You can make a hidden application visible by including a .**SetVisible** command in your script.
- If the last script application command before the **END WITHOBJECT** statement is not a .**FileClose** (for CorelDRAW or PHOTO-PAINT) or .**FileExit** (for Corel VENTURA), the hidden application becomes visible after executing the last command.

Tip

- You can also execute a script by double-clicking on its icon in the Windows Explorer or in a Windows folder.

{button ,AL(`cse_setvisible;draw_setvisible;flow_setvisible;vent_setvisible;pp_setvisible;cad_setvisible;Creating and modifying scripts_cs;Executing_script_files;Executing applications commands in the background;script_files;Script_planning_designing;',0,"Defaultoverview",)} Related Topics

Sample scripts

Your Corel application comes with sample scripts. You can use these scripts as they are, or you can modify them to work better for you. You can also use them as a foundation for creating your own scripts.

The sample scripts are available in the following folders (based on a typical Corel installation):

C:\corel\scripts	The scripts in this folder are not application specific: they do not send instructions to Corel applications.
C:\corel*application*	The scripts in this folder are application specific: they send instructions to Corel applications to perform actions. ***application*** refers to the application's folder. For example, CorelDRAW scripts are saved in the **C:\corel\draw** folder. For a list of the sample scripts included with your Corel application, see the Reference section in your Corel application's online Help.

You can also save your own scripts in the folders mentioned above.

Notes

- Comment statements at the top of each sample script describe what the script can do for you. If a script's first line, second line, or both lines, are REM statements, the comments are displayed in a Corel application's Run Script dialog box if the script is specified. The same two REM statements are also displayed in the status bar if the script is assigned to a tool bar button.
- Unlike script files (or macro files) from other companies, Corel SCRIPT files are text only; they contain no compiled binary component. Before a script is executed, it is compiled internally into a program file.

{button ,AL(`intro_cs;;;;;',0,"Defaultoverview",)} **Related Topics**

Script planning and designing tips

It is a good idea to plan the script before you begin writing. Keep the following criteria in the mind:

- How much of any given process do you want to automate? Will the script be useful for one document, and not of much use in another? Which input boxes and dialog boxes are needed to obtain information while the script is running?
- Building error checking in your scripts prevents unexpected results. For example, you can tell a script to terminate if the intermediate script results are not suitable. You can also keep the user from entering inappropriate information, such as an inappropriate value in a dialog box. Click ⇨ for more information.
- Build and test your scripts in parts. You're apt to make fewer mistakes when you build your scripts in smaller sections. Testing and debugging smaller script sections is easier than trying to trace errors through a large complicated script.
- You should run the scripts you write or edit from the Corel SCRIPT Editor (not from another Corel application) until they run error free. Executing scripts from the Editor lets you take advantage of its testing and debugging features. However, once you are satisfied that a script is running properly, you should run it from a Corel application to save execution time.
- Take some time to lay out the steps a script will perform. Use plain language to describe the problem to be solved and the script solution in remark statements (**REM**) at the beginning of the script. Use remark statements at the beginning of each section to describe what that section will do.
- Use the command recording feature in Corel applications to record the task you want to automate. You can then edit the recording and include programming statements, for example, looping constructs and conditional statements, without worrying about the recorded application commands being syntactically correct. Not every Corel application supports Corel SCRIPT programming and scripts. Click ⇨ for a list of Corel applications that support Corel SCRIPT and recording commands.

{button ,AL(`cs_fund;;;;;',0,"Defaultoverview",)} **Related Topics**

Corel applications supporting Corel SCRIPT and OLE Automation

Application	WITHOBJECT string	Supports
CorelDRAW 6	"CorelDraw.automation.6"	NO
CorelDRAW 7	"CorelDraw.automation.7"	YES
CorelDRAW 8	"CorelDraw.automation.8"	YES
CorelDRAW 9	"CorelDraw.automation.9"	YES
Corel PHOTO-PAINT 6	"CorelPhotoPaint.automation.6"	YES
Corel PHOTO-PAINT 7	"CorelPhotoPaint.automation.7"	YES
Corel PHOTO-PAINT 8	"CorelPhotoPaint.automation.8"	YES
Corel PHOTO-PAINT 9	"CorelPhotoPaint.automation.9"	YES
Corel SCRIPT Editor 7	"CorelScript.automation.7"	NO
Corel SCRIPT Editor 8	"CorelScript.automation.8"	NO
Corel SCRIPT Editor 9	"CorelScript.automation.9"	YES
Corel VENTURA 7	"CorelVentura.automation.7"	YES
Corel VENTURA 8	"CorelVentura.automation.8"	YES

- The **WITHOBJECT** statement is used to direct Corel SCRIPT application commands to the Corel application that the commands are used in.
- Corel SCRIPT is not compatible with the macros and PerfectScript used with Corel WordPerfect Suite applications (Corel WordPerfect, Corel Quattro Pro, and Corel Presentations).
  - All three applications do support OLE Automation. You could use the Corel SCRIPT Editor to create a script that sends instructions to any of these applications using the Corel application's macro or PerfectScript commands.
- In Corel SCRIPT 6, predefined constants were used to specify an application. For example, **DRAW** for **CorelDraw.Automation.6**. This option no longer exists, but you can now use constants defined in the **INCLUDE** files distributed with Corel SCRIPT. See the **INCLUDE** statement for more information.

{button ,AL(`include;with_end_with;;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT file types

The following file types are used in Corel SCRIPT:

Corel SCRIPT scripts (.CSC)

Corel SCRIPT scripts are text files that do not have a compiled executable component. Before a script is executed, it is internally compiled into a binary file, and is recompiled each time the script is run.

Corel SCRIPT include scripts (.CSI)

CSI files are scripts (text files) that define commonly used script components such as constants and procedures. These files can be edited in the Corel SCRIPT Editor, and each Corel application that supports Corel SCRIPT has its own .CSI file. Click ⇨ for more information.

Corel SCRIPT binaries (.CSB)

Corel SCRIPT binaries are created by compiling a script into computer binary format. Binaries can only be executed from the Corel SCRIPT Editor or a Corel application. Click ⇨ for more information.

Corel SCRIPT Executables (.EXE)

Corel SCRIPT Executables are standalone program files (.EXE) which are created by compiling a script into computer binary format. Unlike Corel SCRIPT binaries, executables can be run as a standalone application ⇨ the Corel SCRIPT Editor or a Corel application are not necessary to run an executable. Click
⇨ for more information.

Dynamic Link Libraries (.DLL)

Dynamic link libraries DLLs created with Corel SCRIPT contain only one exportable function. This function, named **RUN**, executes the commands in the script that were used to create the DLL. Click ⇨ for more information.

Corel Add-ons (.CAO)

Add-ons are compiled software modules that can be used to customize and enhance a Corel application. Essentially, Corel Add-ons are special Windows DLLs (with a CAO extension) that have access to the Corel application programming interface (API). Add-ons can be created in Corel SCRIPT or in the programming environment you are most familiar with such as Pascal, FORTRAN, C, or C++. Currently, Corel VENTURA 7 is the only Corel application that supports Corel Add-ons. Click ⇨ for more information.

{button ,AL(`Distributing_cs_exe;;;;;',0,"Defaultoverview",)} **Related Topics**

Corel SCRIPT syntax

Syntax refers to rules that govern the form of script statements, commands, and functions. For example,

```
.FileOpen "C:\file.xxx"
```

opens the FILE.XXX file. The next example,

```
.FileOpen "C:\file.xxx
```

lacks a closing quotation mark. The syntax is not correct and produces an error message. In order for scripts that you type from the keyboard to work properly, script commands and their elements must be arranged in the correct order. To be syntactically correct, each script statement or command must be spelled correctly, and must include all of the required elements and the necessary separators in the correct order.

Script statements, commands and functions consist of three parts: a name, parameters, and separators.

Name

The name indicates which feature the Corel SCRIPT statement, command, or function activates. Sometimes, the name is all that is necessary to perform a complete action. For example, **BEEP** is a complete script statement because Corel SCRIPT does not need any additional information; the statement name itself sounds your computer's default beep sound.

Example: `BEEP`

Parameters

If the statement, command, or function needs more information than is provided by the name alone, parameters are required. The name represents the feature. Parameters represent aspects of the feature you can change, or selections you can make. For example, the **ABS** function requires one parameter that specifies the number you want to return a absolute value for. In the following example, ABS returns the absolute value of -123,

```
My_Number = ABS(-123)
```

For parameters that need numeric information, you can use numbers, variables, constants, functions, operators, or any combination of these, as long as they return a numeric result. The above example can be specified in the following manner:

```
My_Number = ABS(x) 'variable
My_Number = ABS(x - y) 'formula using variables
My_Number = ABS(x > y) 'conditional formula
```

The same type of rules apply to sting parameters. For more specific information about using parameters, see Syntax for application commands and functions.

Separators

Some Corel SCRIPT statements, commands, and functions require several parameters. Parameters generally need to be placed in the correct order, and must be separated properly. Commas (,) are used to separate individual parameters. In other cases, parameters need to be enclosed by parenthesis. Spaces between the command names, parameters, commas, and parenthesis are optional. For more information about the syntax of a specific statement, command, or function, see its topic in this help file (click Contents, Index, and type the name of the command).

Note

Corel SCRIPT is not case-sensitive; that is, when a Corel SCRIPT script is compiled, the names of all constants, variables, functions, and subroutines are converted to uppercase. Therefore, you cannot have a variable called **Left**, for example, because when converted to uppercase, it would conflict with the Corel SCRIPT function called **LEFT**. The following commands are all interpreted the same way by Corel SCRIPT:

- LOOP
- loop
- lOOp

Strings surrounded by double quotation marks are not converted when a script is compiled and executed.

- If you create scripts by recording keystrokes and mouse clicks, the commands are automatically inserted in the correct format.
- From the Corel SCRIPT Editor, you can open the Corel SCRIPT online Help to a function's, command's or statement's syntax reference by placing the insertion point in the keyword you want help for, and pressing F1.

{button ,AL(`intro_cs;cs_case_sensitive;corel_script_editor_basics;;;',0,"Defaultoverview",)} Related Topics

Syntax for application commands and functions

The application commands and functions section in this online Help provides syntax and reference information for all Corel application commands and functions. The following is an example of a typical Corel SCRIPT application command (the Corel VENTURA file export command):

.FileExport .FileName=*string*, .FilterType=*long*, .Markup=*Boolean*

Application command names and functions are always preceded by a period. If they use parameters, the parameters are placed to the right of the application command or function name. Required parameters are noted in bold type, and optional parameters are noted in normal type in the syntax reference.

Application parameters have two parts: the parameter name and the data type (in italics). In this example, **FileName** is the parameter name and **string** is the data type. Data type indicates the type of information that the parameter can pass. For more information about data types, click ➪.

With most Corel SCRIPT application commands, you have options on how to specify parameters. The following examples show two methods to specify the .**FileExport** command:

```
.FileExport .FileName="C:\NEWFILE.RTF", .FilterType=11, .Markup=0
.FileExport "C:\NEWFILE.RTF", 11, 0
```

The first example uses the parameter names and equals signs, while the second example separates parameters with commas. Using the parameter names makes your script more self-documenting, but it isn't as easy to write. Corel application commands are recorded using the second example's formatting. For more information about parameters, click ➪.

You also have the option to mix the parameter specification for a command. For example:

```
.FileExport "C:\NEWFILE.RTF", .FilterType=11, .Markup=0
```

Once you use a parameter name in an application command, the parameters that follow in that command must also use parameter names.

Omitting optional parameters

The following application command's last two parameters are optional:

.RotateObject .Angle=*double*, .Clockwise=*Boolean*, .ObjectName=*string*, .FileNumber=*integer*

In this case, you can specify .**RotateObject** without specifying the third parameter, the fourth parameter, or both, as shown in the examples below (the apostrophe indicates a script comment):

```
.RotateObject 58.3, -1, "poly", 2 'no omitted parameters
.RotateObject 58.3, -1, , 2       'third parameter omitted
.RotateObject 58.3, -1, "poly"    'fourth parameter omitted
.RotateObject 58.3, -1            'third and fourth parameters omitted
```

You can also omit optional parameters when you use the parameter names, as shown in the examples below (these examples replicate the previous example set):

```
.RotateObject .Angle=58.3, .Clockwise=-1, .ObjectName="poly", .FileNumber=2   'no omitted
.RotateObject .Angle=58.3, .Clockwise=-1, .FileNumber=2   'third omitted
.RotateObject .Angle=58.3, .Clockwise=-1, .ObjectName="poly"   'fourth omitted
.RotateObject .Angle=58.3, .Clockwise=-1,  'third and fourth omitted
```

Parameters and data types

Parameters must be of the correct data type; if a string is required, the argument must be a valid string expression, or an error occurs. Numeric parameters are not always as strict in terms of data types. For example, an application command parameter may require a long (a whole number). If you pass the parameter a fractional number instead, an error may occur, or the parameter may accept the fractional number and truncate it.

Note

- Not every Corel application supports Corel SCRIPT programming and script files. Click ➪ for a list of Corel applications that support Corel SCRIPT.
- From the Corel SCRIPT Editor you can open Corel SCRIPT online Help to a function's, command's or statement's syntax reference by placing the insertion point in the keyword you want help for and pressing F1.

{button ,AL(`CS_app_commands_exe;using_variables;Documentation_syntax_conventions;cs_case_sensitive;Corel_S

CRIPT_Editor_Basics;Variable_availability;;',0,"Defaultoverview",)} **Related Topics**

Syntax for programming statements and functions

Programming statements follow the same parameters specification rules as application commands (See Syntax for application commands and functions for more information). However, for the most part, programming statements and functions do not use parameter names. Script parameters specify the variable type by using the data type suffix.

Note

- Parameters must appear in the order specified in the Corel SCRIPT programming statements and functions section in this online Help.
- Parameters must also be separated by commas, as specified in the Corel SCRIPT programming statements and functions section in this online Help.
- From the Corel SCRIPT Editor, you can open Corel SCRIPT online Help to a function's, command's or statement's syntax reference by placing the insertion point in the keyword you want help for, and pressing F1.

{button ,AL(`using_variables;Documentation_syntax_conventions;cs_case_sensitive;Corel_SCRIPT_Editor_Basics;Variable_availability;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT documentation syntax conventions

The Corel SCRIPT programming statements and functions section provides syntax and reference information for each statement and function. Click for a listing of all programming statements and functions.

The Corel SCRIPT application commands and functions section in this online Help provides syntax and reference information for all application commands and functions. Not every Corel application supports Corel SCRIPT programming and script files. Click for a list of Corel applications that support Corel SCRIPT.

The following typographic conventions are used to show the syntax for all Corel SCRIPT programming and application commands.

Syntax convention example	Description
LBOUND, **IF**, **LCASE**	Corel SCRIPT programming statements and functions appear in boldface, all uppercase.
.FileOpen, .**EditCut**	Corel SCRIPT application commands and functions appear in boldface, with the initial letter of each word in uppercase. The command is preceded by a period. Generally, Corel SCRIPT application command names correspond to the command's menu name, preceded by its main menu name. For example, the .**EditCut** command is the complement of a menu command (click Edit, Cut) on an application's menu system.
Required parameters	Required parameters for Corel SCRIPT programming statements and functions appear in boldface.
Optional parameters	Optional parameters for Corel SCRIPT programming statements and functions appear in normal face.
Command parameter	Required parameters for Corel SCRIPT application commands and functions appear in an boldfaced italics.
Command parameter	Optional parameters for Corel SCRIPT application commands and functions appear in italics.
{%|&|!|#|@|$}	Braces and vertical bars surrounding the variable type suffixes indicate a choice between two or more items. The choice is mandatory if the syntax appears in boldface. If the syntax appears in normal typeface, use of the suffix is optional.
{ WHILE | UNTIL }	Braces and vertical bars surrounding statement keywords indicate a choice between two or more items. The choice is mandatory if the syntax appears in boldface. If the syntax appears in normal face, syntax use is optional.
Script examples	Script example statements and functions appear in this font.
'Comments	Script example comments appear with an apostrophe ('). In some cases, the **REM** statement is used instead of the apostrophe.
Description Text	Commands, statements, and parameters in description text appear in boldface.
FILENAME, FOLDERS	Filenames, folders, and paths appear in uppercase letters in description text.

- Click the button for a typical example of a Corel SCRIPT programming statement syntax topic in this online Help.
- Click the button for a typical example of a Corel SCRIPT application command syntax topic in this online Help.

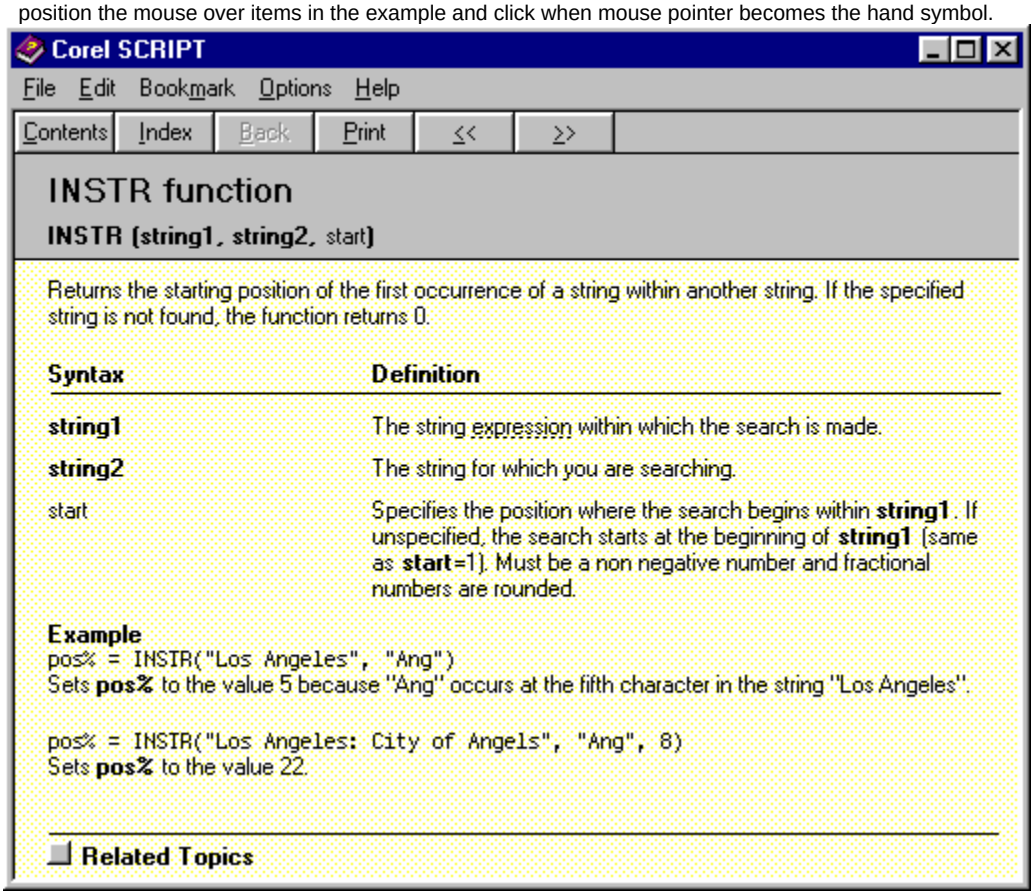
Note

- Corel SCRIPT application command and function parameters must be separated by commas.
- Required parameters are displayed in boldface type, while optional parameters are displayed in normal typeface. Optional parameters normally appear at the end of a command's syntax.

{button ,AL(`cs_syntax_ov;;;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT programming statement syntax example

This is a typical example of a Corel SCRIPT programming statement syntax topic in this online Help. For more information, position the mouse over items in the example and click when mouse pointer becomes the hand symbol.



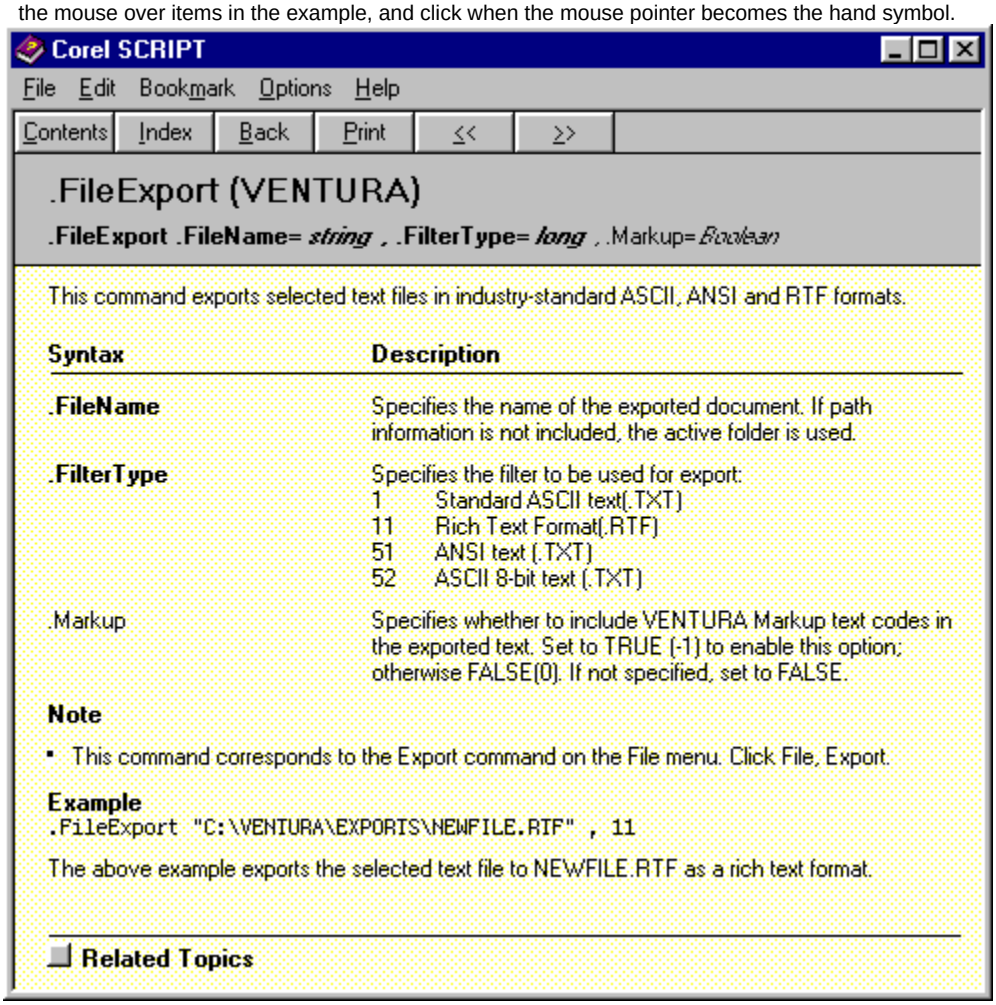
Note

- From the Corel SCRIPT Editor you can open Corel SCRIPT online Help to a functions, command's or statement's syntax reference by placing the insertion point in the keyword you want help for and pressing F1.
- From the Corel SCRIPT Editor you can open Corel SCRIPT online Help to a function's, command's or statement's syntax reference by placing the insertion point in the keyword you want help for and pressing F1.

{button ,AL(`cs_syntax_ov;;;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT application command syntax example

This is a typical example of a Corel SCRIPT application command syntax topic in this online Help. For more information, position the mouse over items in the example, and click when the mouse pointer becomes the hand symbol.



Note

- From the Corel SCRIPT Editor, you can open Corel SCRIPT online Help to a function's, command's or statement's syntax reference by placing the insertion point in the keyword you want help for, and pressing F1.

{button ,AL(`CS_app_commands_exe;cs_syntax_ov;;;;;',0,"Defaultoverview",)} Related Topics

Provides a list of similar commands and functions.

Displays the application command name and the Corel application it is used with. For example DRAW for CorelDRAW, VENTURA for Corel VENTURA, and so on.

Displays the syntax to use with the application command. The syntax that appears to the right of the application command is the command's parameter set. Parameters are used to pass information to the application command. Required parameters are noted in boldfaced type, and optional parameters are noted in normal typeface.

Application parameters have two parts: the parameter name and the data type (in italic typeface). In this example, **FileName** is the parameter name and **string** is the data type. Data type indicates the type of information that the parameter can pass. For more information about data types, click ⇨. With most Corel SCRIPT application commands, you have options on how to specify parameters. The following examples show two methods to specify the .**FileExport** command:

```
.FileExport .FileName="C:\NEWFILE.RTF", .FilterType=11, .Markup=0
.FileExport "C:\NEWFILE.RTF", 11, 0
```

The first example uses the parameter names and equals signs, while the second example separates parameters with commas. Using the parameter names makes your script more self-documenting, but is not as easy to write. Corel application commands are recorded using the second example's formatting.

Displays a description of what the command can do.

Displays a description of the parameters and the information they can pass.

Displays additional information about using the application command. Often, the note indicates how to record the command.

Displays an example of the command.

Displays the programming statement or function name.

Displays the syntax to use with the programming statement or function. The syntax that appears to the right of the programming statement or function name is the command's parameter set. Parameters are used to pass information. Required parameters are noted in boldfaced type, and optional parameters are noted in normal typeface.

Displays a description of what the programming statement or function can do.

Displays a description of the parameters and the information they can pass.

Displays an example of the statement or function.

Provides a list of similar statements and functions.

Compiling and debugging scripts

Compiling scripts

Corel SCRIPT scripts are text files; they do not have a compiled binary component. Before a script is executed, it is compiled internally into a executable file and then run.

What is an executable

A computer's central processing unit (CPU) plays instructions written in machine (computer) language. Machine language consists of binary digits (bits) 0 and 1. For example, the first three letters of the alphabet in binary notation are 1000001, 1000010, and 1000011. The binary result of 4 + 5 is 1001. Because binary digits are hard to work with, English-based programming languages (such as BASIC, Pascal, and C) were created. Programs are written with the help of an editor or word-processor, and saved as source files.

Corel SCRIPT can be considered a dialect of BASIC. Scripts are written with a text editor and saved. Before a script is executed, Corel SCRIPT converts the source file (the script) into binary format and runs it. The binary format is not saved in the script and must be recompiled each time the script is run.

Note

- You can create a compiled version of a script, a Corel SCRIPT Executable. See Creating Corel SCRIPT Executables for more information.

{button ,AL(`comp_debug;;;;;',0,"Defaultoverview",)} Related Topics

Script programming errors and debugging

When you edit a recording or write your own scripts, programming errors can occur. There are three main types of errors that can occur:

- Compilation errors
- Run-time errors
- Logic errors

Notes

- Corel SCRIPT scripts are Windows text files; they do not have a compiled binary component. Before a script is executed, it is compiled internally into a program file, and then run. You can create a compiled version of a script a Corel SCRIPT Executable. See Creating Corel SCRIPT Executables for more information.
- If you create scripts by recording keystrokes and mouse clicks, the commands are automatically inserted in the correct format.
- From the Corel SCRIPT Editor, you can open Corel SCRIPT online Help to a function's, command's or statement's syntax reference by placing the insertion point in the keyword you want help for and pressing F1.
- Scripts that use application commands should be run from the Corel SCRIPT Editor until they run error free. Executing scripts from the Editor allows you to take advantage of its testing and debugging features. However, once you are satisfied a script is running properly, you should run the script from a Corel application for significant time savings.
- If Corel SCRIPT attempts to execute an application's commands, and it is not already running, Corel SCRIPT attempts to start the application in a portion of the computer memory called the *background*. The best way to ensure that all applications running in the background are closed after script execution, is to have the script close all its documents. If your script is not closing the open documents and leaving the application to run in the background, press CTRL+ALT+DEL to open the Close Program dialog box to close the invisible applications.
- In some cases, you can have both a visible and an invisible instance of an application running in the background. In this case, Corel SCRIPT executes the application commands on the application instance which was started first. This can result in confusion if the invisible application was started first, and you expect the Corel SCRIPT application commands to be executed in the visible instance of the application. Press CTRL+ALT+DEL to open the Close Program dialog box to close the instance of the application you want to close.

{button ,AL(`comp_debug;;;;;',0,"Defaultoverview",)} Related Topics

Compilation errors

Corel SCRIPT scripts are text files; they do not have a compiled binary component. Before a script is executed, it is compiled internally into a program file, and then run. Compilation errors prevent Corel SCRIPT from compiling a script into machine instructions. Compilation errors are easy to find since Corel SCRIPT detects them, and notes them in the Compiler Output Window in the Corel SCRIPT Editor.

The most common compilation errors include:

- typographic errors such as misspelling variable names or command names
- missing opening or closing bracket, or parentheses
- missing a required parameter
- missing a corresponding closing statement, for example, omitting the FOR statement when using the FOR...NEXT construct
- incorrect use of a statement or function

The compiler identifies and reports a maximum of ten errors. Once the compiler finds eleven errors, it stops. You must correct the errors, and re-compile. The compiler also suggests solutions for the script writer to verify.

For more information about other errors, see Error codes.

Tip

- The Check Syntax command (click Debug, Check Syntax) in the Corel SCRIPT Editor searches for all syntax errors.

{button ,AL(`comp_debug;;;;;',0,"Defaultoverview",)} Related Topics

Run-time errors

Run-time errors occur when a script is run, or executed. These errors are generated when scripts produce bad or illegal values, or try to run an impossible operation. You can design your scripts with run-time errors in mind. For example, an UNABLE TO DELETE FILE run-time error might occur when you try to remove a file. Since you may not know the restrictions on the file you're trying to remove, you could write an error handling routine to trap for this possibility and have your script act accordingly. See Error Codes or the **ON ERROR** command for more information.

The most common run-time errors include:

- division by 0
- variable type mismatch
- file access errors
- variable overflow

Tip

- Use the **MESSAGE** statement to display a built-in Corel SCRIPT message dialog box. A message box stops script execution and can be used to display the value of variables.

{button ,AL(`comp_debug;;;;;',0,"Defaultoverview",)} Related Topics

Logic errors

Logic errors are the hardest to find; the only indication of a logic error may be a bad value, or an unexpected result. The Corel SCRIPT Editor cannot tell you when a logic error is present, so it is up to you to test for and find these problems. The Corel SCRIPT Editor provides powerful debugging tools to help you debug scripts:

- run scripts in slow motion (step-by-step), pausing as long as you want between steps
- set breakpoints to pause a script when it reaches a specified line in the script
- run scripts at full speed until reaching a breakpoint, then either continue in slow motion, or at full speed until the next breakpoint
- monitor, or trace, changes to specific variables as a script runs

For more information about these tools, see Corel SCRIPT Editor debugging features .

Tip

- Use the **MESSAGE** statement to display a built-in Corel SCRIPT message dialog box. A message box stops script execution and can be used to display the value of variables.

{button ,AL(`comp_debug;;;;;',0,"Defaultoverview",)} Related Topics

Advanced features

Using Dynamic Link Libraries

Corel SCRIPT can call functions and subroutines in Dynamic Link Libraries (DLLs), such as those supplied with Windows, other applications' DLLs, or any custom DLL files. To find out how to use functions in DLLs, you need specific technical reference material. For example, to use the Windows DLLs, you need the *Windows Software Development Kit*.

Before you can use a DLL function or subroutine, you must declare the function using the **DECLARE...LIB** statement. See the statement's reference for more information and an example.

Warning

- You should save or back up essential files and programs before using functions and subroutines in DLL files. Passing an invalid argument to a function can result in a Windows General Protection Fault or unstable system behavior.

Note

- The advanced Corel SCRIPT programming feature, described above, is intended for experienced Windows programmers, and not for beginner script writers.

{button ,AL(`declare_lib;getapphandle;getwinhandle;;;',0,"Defaultoverview",)} Related Topics

Creating Corel SCRIPT Executables, Binaries, DLLs, and Corel Add-ons

Corel SCRIPT scripts are text files that do not have a compiled executable component. Before a script is executed, it is internally compiled into a binary file, and is re-compiled each time the script is run. To save time, compile your scripts into Corel SCRIPT Executables, Corel SCRIPT Binaries or Dynamic Linked Libraries (DLLs). Compiling your scripts into Executables, Binaries, or DLLs, not only speeds up their run-time, but also allows you to hide the programming code.

Corel SCRIPT Executables

Corel SCRIPT Executables are standalone program files (.EXE) which are created by compiling a script into computer binary format.

Corel SCRIPT Binaries

Like Corel SCRIPT Executables, Binaries (.CSB) are created by compiling a script into computer binary format. However, Binaries can only be executed from the Corel SCRIPT Editor or a Corel application. For example, if you have a script that applies a fountain fill to objects in CorelDRAW 7, you could convert the script into a Binary. When you want to apply the fountain fill to an object, simply drag the Binary onto the selected object within CorelDRAW.

Dynamic Linked Libraries

DLLs created with Corel SCRIPT contain only one exportable function. This function, named **RUN**, executes the commands in the script that was used to create the DLL.

Corel Add-ons

You can use Corel SCRIPT to add your own compiled software modules to Corel applications so that you can customize and enhance their features. These software modules, called Corel Add-ons, are special Windows DLLs (with a CAO extension) that have access to the Corel application programming interface (API). Currently, Corel VENTURA 7 is the only Corel application that supports Corel Add-ons.

Speed is the major advantage of using an Add-on rather than a script or executable. Other advantages of using Corel Add-ons include the following:

- They can contain routines that can be called by scripts or executables.
- An Add-on function can be called the same way you call a routine from a regular DLL.
- Corel Add-ons can be written in the programming environment you are most familiar with such as Pascal, FORTRAN, C, or C++.
- The Corel application API gives your programming language full access to all Corel SCRIPT application commands.

Like DLLs created with Corel SCRIPT, Corel Add-ons created with Corel SCRIPT contain only one function (**RUN**). This function executes the commands in the script that was used to create the Add-on. If you create a Corel Add-on in a programming environment other than Corel SCRIPT, you are not limited to a single function. See Developing Corel Add-ons for VENTURA for information about writing your own Add-ons in languages other than Corel SCRIPT.

Executing Corel SCRIPT Executables, Binaries, DLLs, and Corel Add-ons

Corel SCRIPT Executables or DLLs can run without having a Corel application open or running. Both, however, as well as Corel SCRIPT Binaries and Corel Add-ons created with Corel SCRIPT, require the Corel SCRIPT run-time interpreter to be installed on the user's system when run. The run-time interpreter is a Dynamic Linked Library named SCINT**xx**.DLL, where **xx** indicates the Corel SCRIPT major version number for example, SCPINT70.DLL for Corel SCRIPT version 7.0. The run-time interpreter contains the instructions that tell your computer how to execute the Corel SCRIPT programming instructions in the Executable, Binary, DLL, or Add-on.

Click for a list of locations on a user's system where a Corel SCRIPT Executable, Binary, DLL, or Add-on searches for the run-time interpreter.

Corel SCRIPT versions

For a script, Executable, Binary, DLL, or Corel Add-on created with Corel SCRIPT to run, the major version numbers for Corel SCRIPT (the compiler) and the Corel SCRIPT run-time interpreter (SCINT**xx**.DLL) must be the same or else an error will occur. A difference in the minor version numbers will not cause an error. See the **GETVERSION** function for information about determining the Corel SCRIPT or run-time interpreter version numbers.

Note

- You can edit and change a script after it has been used to compile a Corel SCRIPT Executable, Binary, or DLL without affecting the compiled file.
- Although you can't assign a Corel SCRIPT Executable directly to a toolbar in a Corel application, you can assign it to the Corel Application Launcher toolbar button . Click for more information.

- See the Corel Corporation World Wide Web home page (www.corel.com) for updates to the Corel SCRIPT run-time interpreter (SCINT**xx**.DLL).

{button ,AL(`addresbmp;Distributing_cs_exe ;Using_Dynamic_Link_Libraries ;GETVERSION;GETSCRIPTFOLDER;;;',0,"Defaultoverview",)} **Related Topics**

Distributing Corel SCRIPT Executables, Binaries, DLLs, and Corel Add-ons

You can freely distribute the Corel SCRIPT Executables, Binaries, DLLs, and Corel Add-ons you create along with the Corel SCRIPT run-time interpreter (SCINT**xx**.DLL). Provided:

(a) You do not alter or modify the run-time interpreter in any manner.

(b) You do not use the logo, name, or trademarks of Corel or its licensors.

(c) You include a valid copyright notice.

(d) You agree to indemnify Corel and its licensors against any claims or lawsuits arising from your use or distribution of the Corel SCRIPT Executables, Binaries, DLLs, or Corel Adds-on you distribute.

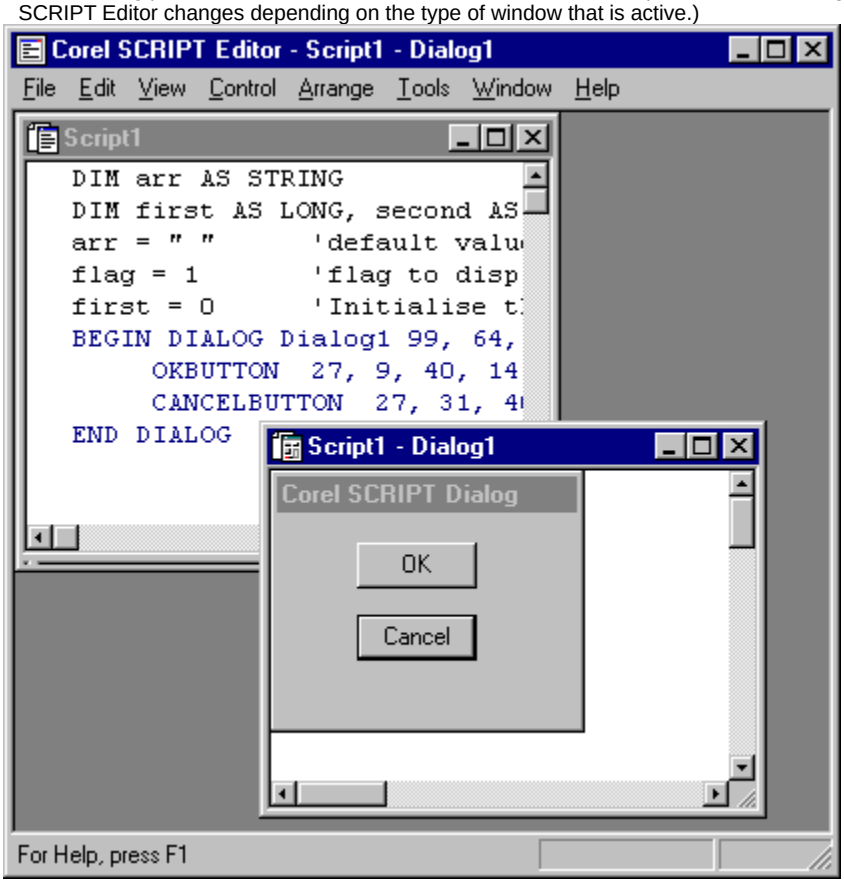
COREL MAKES NO WARRANTY, EXPRESS OR IMPLIED, REGARDING THE USE OF COREL SCRIPT EXECUTABLES, BINARIES, DLLS, OR COREL ADD-ONS, NOR DOES COREL ASSUME ANY OBLIGATION OR LIABILITY FOR THEIR USE.

{button ,AL(`addresbmp;Creating Corel SCRIPT Executables;Distributing_cs_exe ;Using_Dynamic_Link_Libraries ;GETVERSION;GETSCRIPTFOLDER;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT Editor

The Corel SCRIPT Editor creates and edits Corel SCRIPT scripts and custom dialog boxes. Working with a script is similar to working with a text editor or a word processor, while working with custom dialog boxes is similar to using a drawing or painting application. Because creating and editing both scripts and dialog boxes require a unique set of commands, scripts and dialog boxes have their own separate document windows within the Corel SCRIPT Editor; script windows and dialog windows, respectively.

The following picture of the Corel SCRIPT Editor shows both a script window and dialog window. (The menu in the Corel SCRIPT Editor changes depending on the type of window that is active.)



Though script windows and dialog windows are separate, they are not independent. A dialog window can only be opened from an active script window.

For more information about Corel SCRIPT Editor windows, see Script windows and Dialog windows.

Note

- The Corel SCRIPT Editor is a stand-alone program included with Corel applications. If you didn't install the Editor when you installed the Corel applications, you can run the Corel setup program again to install it.
- In Corel SCRIPT version 7.0 you can use Corel SCRIPT application commands and functions to create scripts that can be used to create and modify other scripts. With the addition of customization features to the Corel SCRIPT Editor, the scripting commands give you almost total control over the Corel SCRIPT Editor. Click for a list of Corel SCRIPT application commands for the Corel SCRIPT Editor.

{button ,AL(`Editor_ole;cse_reference;Corel_SCRIPT_Dialog_Editor;Dialog windows and the Corel SCRIPT Editor;;;',0,"Defaultoverview",)} Related Topics

Script windows and the Corel SCRIPT Editor

The Corel SCRIPT Editor included with your Corel application is used to create and modify scripts. Since Corel SCRIPT scripts are Windows text files, the Corel SCRIPT Editor has many of the features of a standard text editor as well as specialized programming features to help you test, debug, modify, and run all or part of a script.

Script windows

You can have more than one script open in the Corel SCRIPT Editor at a time. Each script is in a separate script window and its file name is noted in the window border. The script windows can be arranged in a variety of layouts. Each script window contains a compiler output window and a watch window.

A script's syntax errors are displayed in the Compiler Output window after it has been run or checked for syntax errors. Double-clicking on an error message sends the insertion point to the line containing the error. The line with the error has the  symbol in its left margin after it has been double-clicked. The Compiler Output window is part of a script window and is displayed at the bottom of the window. This window can also be hidden or resized.

Like the Compiler Output window, the Watch window is part of a script window and is displayed at the bottom of the window. The Watch window is used to monitor the value of variables in the script during a debugging session. Each variable being watched displays its current value and the procedure (Main, a function, or a subroutine) where it is found.

Note

- Unlike scripts (or macro files) from other companies, Corel SCRIPT scripts are text only; there is no compiled binary component in a script. Before a script is executed, it is compiled internally into a program file.
- Scripts are saved with the extension .CSC (for Corel SCRIPT) in the SCRIPTS folder, by default. If you use Corel VENTURA 7, scripts can be embedded into a VENTURA document as an internal script.

{button ,AL(`Trappable_error_codes;Script_programming_errors;cse_reference;Corel_SCRIPT_Dialog_Editor;Dialog windows and the Corel SCRIPT Editor;;;',0,"Defaultoverview",)} Related Topics

Script basics

When you create or edit a script using the Corel SCRIPT Editor, you should keep the following in mind:

- Each line can hold up to 1024 characters, including spaces. You can continue script commands and functions over lines by using double backslashes (\\) to break lines.
- As you reach the right edge of the script window with text, the window scrolls to the right as needed. Text never wraps to the next line.
- Each line in a Corel SCRIPT script can contain more than one statement or command. Multiple statements on a line are separated with a colon (:).
- Each line in a Corel SCRIPT script must be followed by a hard return. A hard return is inserted when you press ENTER.
- Use tabs, blank lines, and multiple spaces to format your script to make it easier to read. Tabs, blank lines, and multiple spaces are ignored during script playback.
- Use the arrow keys or the mouse to move the insertion point in the script window. Press the CTRL key with the left and right arrow keys to move from word to word.
- An indicator at the bottom of the script window shows the current line number. This can help you find and fix errors.
- Like most other Windows text editors, you can use the Corel SCRIPT Editor to insert and delete text in script windows. The editor also has cut, copy, and paste features.
- You can have more than one script open in the Corel SCRIPT Editor at a time in a separate script window. Use the commands on the Window menu in the Editor to arrange the script windows or to switch to a different window.

{button ,AL(`cse_reference;;;;;',0,"Defaultoverview",)} **Related Topics**

Formatting a script

If you want to improve the readability of a script, you can format it to include tabs, spaces, and blank lines. Formatting the script will not affect how it works. Formatted scripts also help you to find and fix errors. Here are some tips to make your scripts easier to read:

- Use remarks statements to document your scripts. At the beginning of the script, describe what the script does, how to use it, and give examples of its usage. See **REM** for more information about using remarks.
- Give your scripts meaningful names. You are not limited to the DOS file standard of 8 characters and 3 characters. For example, instead of using **gbo.csc,** use **Gold Bar Object.csc**. It is more descriptive and, six months after writing the script, you will still remember what it does.
- Give your variables meaningful names. You should also capitalize each word after the first or use the underscore character to separate words. While **x**, **y**, and **z** may sound like good variable names, they don't reveal anything about what is supposed to be stored in the variable. An example of better names include **userMaleName**, **character_Total**, **numberOfPicas** and **typeArray**.
- Group commands related to a certain function in the same area, and separate the groups using blank lines. For example:

```
'Creates a new file
.FileNew 360, 504, 1, 300, 300, 0, 0, 0, -1, -1, -1, -1, 255, 255, 255, 0

'next 3 lines set default settings
.SetPaintColor 5, 0, 0, 0, 0
.PenSettings 17, 10, 0, 0, 0, 0, 0
.ShapeSettings 0, 2, 10, 0, 0

'next 3 lines draw a line
.StartDraw 91, 441
.ContinueDraw 213, 214
.EndDraw

'saves the file
.FileSave "C:\myfolder\example.CPT", 1792, 0
```

- When a set of statements lies within a functional area, indent the set to show that they form a subset. For example, with a **FOR...NEXT** structure, indent the lines after the **FOR** statement.

```
FOR a% = 1 To 4
    [statements]
    FOR b% = 1 To 4
        [statements]
        FOR c% = 1 To 4
            [statements]
        NEXT c%
    NEXT b%
NEXT a%
```

- It is a generally accepted programming convention to put declaration statements at the beginning of a procedure.
- Define any frequently used numbers or strings as constants. This way, if the value of the constant changes, you need only to change it at the top of the script, rather than throughout the script, everywhere it is used. Constants and variables should be defined at the beginning of the procedure in which they are used.
- Limit the use of the **GOTO** statement. When you use the **GOTO**, make sure that you don't create spaghetti code scripts that are impossible to trace through, because they repeatedly jump from one area of the script to another.
- Your function and subroutines procedures should be self-contained. A variable required only within a procedure should be a local or static variable. Following this advice can make your procedures more modular, enabling you to copy them to other scripts with limited customization.

{button ,AL(`cs_fund;;;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT Editor debugging features

When you run a script, it may not always perform the way you expect. A script (or program) that does not work correctly is said to have a "bug" in it. The act of finding and correcting these problems is what is traditionally called "debugging." While some mistakes and typographic errors are often easily spotted by looking carefully at a script, some bugs are harder to find. In order to help you in your search, the Corel SCRIPT Editor includes a full set of debugging tools. You can use the following debugging tools in the Corel SCRIPT Editor.

Running a script
An easy way to see if your script has any errors is to run it. If the script does not contain any errors, it executes to the last line. When you run a script that contains programming errors, playback is aborted at the first instance of an error. The error is noted in the Compiler Output window of the Corel SCRIPT Editor. You can also set an option to check for variables that have not been initialized when you run the script.

Checking script syntax
You can check the syntax of each line in a script without running the script. Common syntax errors include misspelled commands, missing operators, and missing punctuation. If errors are found, error messages appear in the Compiler Output window.

Logic errors are the hardest to find and the only indication of a logic error may be a bad value or an unexpected result. The Corel SCRIPT Editor cannot tell you when a logic error is present, therefore it is up to you to test for and find these problems. To help you with this job, the Corel SCRIPT Editor provides the following tools to help you step through a script more carefully, tracking the values of variables and following the flow of execution:

Executing individual lines in a script
You can run individual script lines using the following commands on the Debug menu in the Corel SCRIPT Editor.

Run to Cursor	Executes the script in the active script window to the position of the insertion point. Using the Run to Cursor command is similar to using a breakpoint.
Step Into	Executes a script line by line. The Step Into command also steps into functions and subroutines to execute them line by line.
Step Over	Executes a script line by line. The Step Over command executes an entire procedure (a function or a subroutine) before stopping.
Step Out	Executes the remaining lines in a function or subroutine, returns, and stops at the line after the procedure call.

Using a watch
Stepping through a script to watch the flow of execution can tell you a lot about how a script is performing. It is, however, often just as important to look at the contents of variables as a script runs to see what values they contain. You can monitor a variable's value during script execution with a Quickwatch or the Watch window.

Using breakpoints
When debugging long or complex scripts, it may be difficult to work with the step commands. If the script is long, it's better to mark one or more lines where execution will stop to let you check how things are going. This is done by setting breakpoints.

Using Call Stack window
While stepping through a script, you can use the Call Stack window to display a listing of the active functions or subroutines. The functions and subroutines are listed in order of the most recently used. The first function/subroutine listed is the function/subroutine which is currently running. You can use the Call Stack window to send the cursor to the script position where the active function/subroutine was called.

Help
If you're having trouble understanding the syntax of a Corel SCRIPT, you can get reference information from the help file.

{button ,AL(`cse_reference;debugging_scripts;common_programming_errors;;script_errors;;',0,"Defaultoverview",)}
Related Topics

Corel SCRIPT Editor menu shortcut keys

	Script window active	Dialog window active
File Menu		
CTRL+N	New	New
CTRL+O	Open	Open
CTRL+S	Save	Save
CTRL+P	Print	
Edit Menu		
CTRL+Z	Undo	
CTRL+Y	Redo	
CTRL+X	Cut	Cut
CTRL+C	Copy	Copy
CTRL+V	Paste	Paste
DEL	Delete	Delete
CTRL+D		Duplicate
CTRL+SHIFT+C	Comment	
CTRL+SHIFT+U	UnComment	
ALT+ENTER		Attributes
Search Menu		
CTRL+F	Find	
CTRL+R	Replace	
CTRL+G	Go To Line	
F4	Next Error	
SHIFT+F4	Previous Error	
View Menu		
ALT+1	Watch Window	
ALT+2	Compiler Output Window	
CTRL+T		Test Dialog
SHIFT+CTRL+G		Grid Settings
Debug Menu		
F5	Run	
CTRL+SHIFT+F5	Restart	
SHIFT+F5	Reset	
F8	Step Into	
CTRL+SHIFT+F8	Step Over	
SHIFT+F8	Step Out	
CTRL+F10	Run To Cursor	
CTRL+F5	Execute	
ALT+F5	Check Syntax	
CTRL+W	Quick Watch	
F9	Toggle Breakpoint	
Tools Menu		
F2	Dialog	
SHIFT+F2		Test Dialog
CTRL+SHIFT+F7		Grid Settings
CTRL+L	Call Stack	
Help Menu		
F1	Help Topics	

SHIFT+F1	What's This?	What's This?

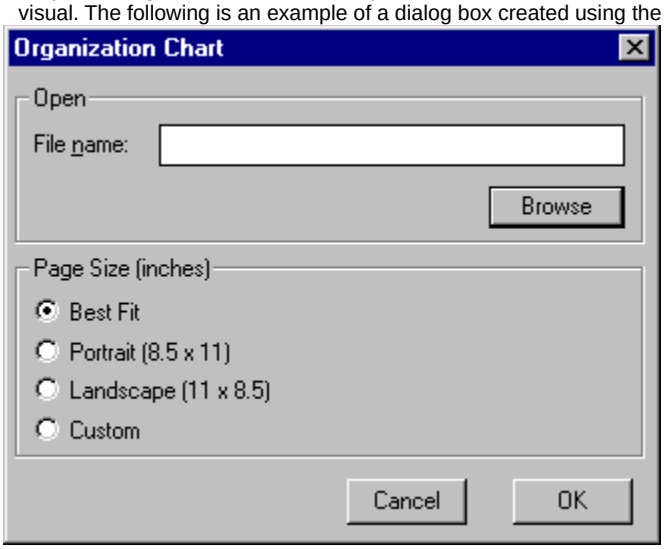
{button ,AL(`Corel_SCRIPT_Editor;Corel_SCRIPT_Editor_Basics;Corel_SCRIPT_Editor_windows;;;',0,"Defaultoverview" ,)} Related Topics

Using the Corel SCRIPT Editor to create dialog boxes

Many times you need to get information from the user before your script performs a desired action. For simple information, you can use the Corel SCRIPT **INPUTBOX** statement to get a string from the user returned to a running script. If you want to provide the user with options and more complex information, such as a list of choices, you can use a dialog box in your script.

Along with creating and modifying Corel SCRIPT scripts, the Corel SCRIPT Editor can also be used to create and edit custom dialog boxes that can be used in a script. Working with dialog boxes in the Editor is similar to using a drawing or painting application: Dialog controls are graphic objects which can be inserted, moved, resized, and aligned in a dialog box.

Working with dialog boxes in the Corel SCRIPT Editor takes the place of creating and editing Corel SCRIPT statements in a script. A dialog box and its controls represent Corel SCRIPT statements; therefore the process of creating and editing them is visual. The following is an example of a dialog box created using the Corel SCRIPT Editor.



{button ,AL(`csde;Corel_SCRIPT_Dialog_Tips;;;;',0,"Defaultoverview",)} **Related Topics**

Dialog windows and the Corel SCRIPT Editor

Dialog boxes are edited in dialog windows. A dialog window can only be created and opened from a script window, which is used to modify scripts. A relationship exists between these two windows. The dialog window is considered a child of the parent script window. When the script window closes, the dialog window closes as well; however, the dialog window can be closed on its own.

A dialog window is opened by placing the insertion point within a script window and clicking Tools, Dialog. Depending where the insertion point is placed, the command either creates a new dialog box or opens an existing dialog. If the insertion point is placed in an existing dialog box definition, that dialog definition is sent to a dialog window to be displayed as a dialog box. If the insertion point is not placed in a dialog definition, a new dialog box is created in a dialog window and the **BEGIN DIALOG** and **END DIALOG** statements are inserted into new lines immediately below the line that contained the insertion point in the script window.

In either case, as you edit the dialog box in the dialog window, the script that launched the dialog window reflects the changes you make to the dialog box. For example, if you add a check box to the dialog box, a CHECKBOX statement is added to the script. You cannot edit a dialog box definition in a script window while it is open in a dialog window. However, you can still continue to edit other statements in the script, including dialog box definition statements, that have not be launched to dialog windows. While a dialog box definition is open in a dialog window, it appears in a different colored font than the rest of the script that launched it.

The following picture of the Corel SCRIPT Editor shows a script window which has launched a dialog window:

Once you have finished editing a dialog box in dialog window, close the window and the dialog box definition becomes available to be edited in a script window.

Note

- The Corel SCRIPT Editor is a stand-alone program included with Corel applications. If you didn't install the Editor when you installed the Corel applications, you can run the Corel setup program again to install it.

{button ,AL(`csde;Corel_SCRIPT_Dialog_Tips;;;;',0,"Defaultoverview",)} **Related Topics**

Working with dialog windows

The easiest way to work in dialog windows is to use the mouse, instead of the keyboard, to create and edit dialog boxes and their controls, much the same way you do with a drawing or painting program. The mouse pointer in a dialog window has two states: the Selector state (default status) and the Control state.

In the Selector state, the mouse pointer appears as and is used to select, resize, and drag and drop dialog controls. You can also select, move, and resize dialog boxes. By default, the mouse pointer is in the Selector state, and when not in the Selector state can be reset by clicking

from the tool bar or clicking Control, Selector from the menu system.

In the Control state the mouse pointer appears as and is used to create dialog controls. The mouse pointer can only be set to a Control state by clicking any of the dialog controls in the Control ribbon bar or any control under the Control menu.

Note

- The mouse pointer can only appear in a Control state when it is positioned over a dialog box in an active dialog window.

{button ,AL(`csde;;;;;',0,"Defaultoverview",)} **Related Topics**

Dialog window features in the Corel SCRIPT Editor

- Each time you create new dialog box in the Corel SCRIPT Editor, a dialog window opens containing a template of a dialog box. The template is a dialog box without any controls placed in it. From the window you can resize the dialog box, add and edit controls, and move the dialog box.
- When you move a dialog box within a dialog window, you are actually changing the dialog box's placement on the screen when it is run in a script. Its placement changes because a dialog window represents your computer screen, and moving the dialog box changes its horizontal and vertical position attribute. For example, if the dialog box is placed in the bottom-right corner of its dialog window, it will appear on the bottom-right corner of a computer screen when it is run in a script.
- You can have more than one dialog window open, but each window can only hold one dialog box. Having more than one window open enables you to copy dialog boxes and controls between windows. You can also copy and paste controls from dialog windows to scripts as dialog definition statements.
- In addition to creating and editing dialog boxes, you can also test a dialog box's functionality from a dialog window.

Note

- You can save a dialog box as a set of Corel SCRIPT statements. These statements can be used in other scripts by clicking File, Export. However, on their own, these statements cannot perform a task in a script.

{button ,AL(`csde;;;;;',0,"Defaultoverview",)} **Related Topics**

Inserting and deleting dialog controls

Inserting

To insert dialog controls into a dialog box, the mouse pointer must be in the Control state. To activate the Control state, activate a dialog window and select a control from the Control menu or from the Control ribbon bar. Once in the Control state, the selected control can be added to the dialog box by clicking in the dialog box where you want the top-left corner of the selected control to appear. Each control has a default size setting. You can also use the click and drag method to insert controls in a dialog box.

After the control has been inserted, the mouse pointer reverts back to the Selector state. You can insert multiple controls of the same type by holding down the CTRL key as you select a control from the Control menu or the Control ribbon bar. The mouse pointer remains in the Control state until you press ESC or select the Selector Tool from the menu or ribbon.

Along with a default size setting, most controls have default text and value attributes when inserted in a dialog box. The text labels and values are updated for each instance of the control inserted in a dialog box. For example, the first push button inserted takes the text **PushButton1**. The second push button takes on the text **PushButton2**, and so on.

Deleting

Controls are not permanent fixtures in a dialog box and can be deleted if they are not required.

{button ,AL(`csde;Dialog_controls_summary;Corel_SCRIPT_Dialog_Tips;Dialog_box_conventions;;',0,"Defaultoverview",)} Related Topics

Editing dialog boxes and controls

Dialog boxes and their controls can be moved, copied, deleted, resized, and their attributes can be edited through a variety of methods.

Moving and Resizing

By selecting a dialog box or control, you can move it or resize it using the mouse. Alternatively, opening a dialog box's or control's attribute box enables you to set position and size attributes.

Cut, Copy, Paste and Duplicate

The standard Windows Cut, Copy and Paste commands also function in a dialog window for dialog boxes and their controls. Additionally, the Duplicate command can help speed up your dialog box editing.

Label and identifier editing

You can edit a dialog box's or control's label and identifier setting by opening its attributes dialog box and editing the settings.

Snap to Grid

If you're moving and resizing controls in a dialog box using the mouse, turning on Snap to Grid can help to align and position controls accurately. It also allows you to draw precisely sized controls. The grid is not displayed during a script run.

Note

- When a dialog box or a control is selected, the Editor displays its attributes in the status bar, Properties Bar, or both (if enabled). The status bar also displays the mouse pointer position when it is positioned over a dialog box.

{button ,AL(`csde;Dialog_controls_summary;Dialog_box_conventions;;;',0,"Defaultoverview",)} Related Topics

Aligning and distributing dialog controls

Dialog boxes that are arranged symmetrically, centered, and have the controls aligned are easier to read and understand. The Corel SCRIPT Editor provides a complete set of tools to help you refine your dialog boxes. Using the tools you can:

- Align controls along an edge
- Distribute controls evenly
- Center controls in a dialog box
- Make controls the same size
- Size a control to its label

Snap to Grid

If you're moving and resizing controls in a dialog box using the mouse, turning on Snap to Grid can help to align and position controls accurately. It also allows you to draw precisely sized controls. The grid is not displayed during a script execution.

Note

- When a dialog box or a control is selected, information about its attributes is displayed in the status bar. When the mouse pointer is over a dialog box, the status bar also displays information about the pointer's position.

{button ,AL(`csde;Dialog_controls_summary;Corel_SCRIPT_Dialog_Tips;Dialog_box_conventions;;',0,"Defaultoverview",)} Related Topics

Using dialog units to size and position dialog boxes and controls

Sizing dialog boxes and controls

Every dialog box and dialog box control has a width and height attribute, expressed in dialog units. For width, a dialog unit is 1/4 the average width of the Corel system font. For height, a dialog unit is 1/8 the average height of the Corel system font. In other words, a dialog unit for width and height are practically equal because, on average, the height of the Corel system font is twice its width (1/8 x 2=1/4). Creating a dialog box that is 200 units (width) by 200 units (height) results in a dialog box that is a square or very close to a square.

Positioning dialog boxes and controls

Dialog controls also have attributes which hold position measurements in a dialog box. A control's vertical position is measured in width dialog units from the inside of the dialog box's left border to the left side of the control. A control's horizontal position is measured in height dialog units from the bottom of the dialog box's title bar to the top of the control.

A dialog box's position during a script run is also set with attributes which hold dialog unit measurements. The position of the left border, with respect to the left side of the monitor's display area, is measured in width dialog units. The position of the top border, with respect to the top of the monitor's display area, is measured in height dialog units.

A dialog box can be centered on the screen by either omitting the position attribute parameters in the **BEGIN DIALOG** statement, or by enabling the Center Dialog check box in the attributes dialog box in a dialog window.

When you create a new dialog box in the Editor, by default the dialog is placed in the top-left corner of the dialog window with the Center Dialog attribute enabled. If the Center Dialog attribute is disabled, the dialog box is displayed in the monitor's top-left corner when run in a script because the dialog window is a representation of your monitor. When you move the dialog box within the dialog window, you are actually changing the dialog box's screen placement when it is run in a script. If you move the dialog box to the bottom-right corner of its dialog window, it will appear on the bottom-right corner of a monitor when run it us in a script.

Notes

- The Corel system font cannot be changed.
- In some cases, changing the screen resolution will change a dialog box's appearance. If your dialog boxes will be used on a variety of screens at different resolution settings, you should test them at each setting to ensure they will be displayed properly.

{button ,AL(`csde;Dialog_controls_summary;Corel_SCRIPT_Dialog_Tips;Dialog_box_conventions;;',0,"Defaultoverview",)} Related Topics

Default control sizes and labels

The following table lists each control's default width, height, and label settings, where applicable.

Control	Width	Height	Label
Bitmap button	46	14	N/A
Cancel button	40	14	Cancel
Check box	50	10	CheckBoxN
Combo box	50	42	N/A
Drop-down combo box	50	42	N/A
Drop-down list box	50	42	N/A
Group box	50	40	N/A
Help button	40	14	Help
Horizontal slider	50	16	N/A
Image	40	40	N/A
Image List Box	50	42	N/A
List box	50	42	N/A
OK button	40	14	OK
Option button	58	10	OptionButtonN
Progress Indicator	50	8	N/A
Push button	46	14	PushButtonN
Spin control	50	12	N/A
Text	50	8	TextN
Text box	50	13	N/A
Vertical slider	16	45	N/A

Note

- N is the Nth occurrence of the control in the dialog box.

{button ,AL(`csde;Dialog_controls_summary;Dialog_box_conventions;Corel_SCRIPT_dialog_controls;;',0,"Defaultoverview",)} Related Topics

Testing dialog boxes

You can test the dialog boxes you create to confirm that they meet your requirements and function properly in a dialog window. It is often easier to test a dialog box in a dialog window than by running it in a script.

To test a dialog box, set the dialog window to test mode. In test mode you can confirm the following dialog box features:

- tab order within the dialog box
- shortcut keys are operational
- drop-down list box openings

Notes

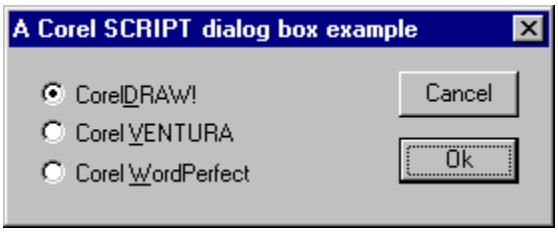
- You cannot edit a dialog box in test mode.
- The following controls are filled with place holders in test mode:
  - list boxes
  - drop-down list boxes
  - combo boxes
  - drop-down combo boxes
  - image boxes
  - bitmap buttons
  - spin controls
- The place holders give you a better idea than an empty control of what the dialog box will actually look like when it is run.

{button ,AL(`csde;Corel_SCRIPT_Dialog_Tips;Dialog_box_conventions;;;',0,"Defaultoverview",)} **Related Topics**

Corel SCRIPT and dialog boxes

Dialog boxes are created using the Corel SCRIPT language. The Corel SCRIPT language features a full set of programming statements to produce dialog boxes which incorporate sophisticated Windows options and features. For example, the Corel SCRIPT statements below create the following dialog box:

```
BEGIN DIALOG Dialog1 55, 10, 180, 53, "A Corel SCRIPT dialog box example"
    OKBUTTON 130, 27, 40, 14
    CANCELBUTTON 130, 7, 40, 14
    OPTIONGROUP ogroup%
        OPTIONBUTTON 10, 9, 90, 10, "Corel&DRAW!"
        OPTIONBUTTON 10, 21, 90, 10, "Corel &VENTURA"
        OPTIONBUTTON 10, 33, 90, 10, "Corel &WordPerfect"
END DIALOG
```



The first and last statements initialize and end a dialog box's definition as well as specifying the dialog box's screen location and size attributes. Each indented statement specifies a dialog control, and the parameters specify the control's dialog box location, size, and other attributes

There are two methods for creating the Corel SCRIPT statements used to produce a dialog box. The first method is to use the Corel SCRIPT Editor and type in the dialog statements. This can prove to be a time-consuming option because each statement's parameters are particular and it is difficult to visualize the dialog box based on coordinate positions.

The second method is to use the Corel SCRIPT Editor to draw your dialog box. The dialog box, and the items in it, are graphical representations of Corel SCRIPT statements. With the Corel SCRIPT Editor you can create or edit a dialog box in a few steps.

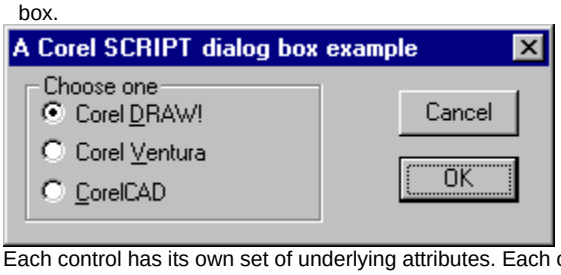
Note

- The Corel SCRIPT Editor is a stand-alone program included with Corel applications. If you didn't install the Editor when you installed the Corel applications, you can run the Corel setup program again to install it.

{button ,AL(`Static and dynamic dialog boxes;Corel_SCRIPT_dialog_controls;Corel_SCRIPT_Dialog_Editor;Dialog windows and the Corel SCRIPT Edito;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT dialog controls

A Corel SCRIPT dialog box is comprised of a border that encloses graphic elements called dialog controls. By using the mouse and keyboard, you can interact with the controls by selecting options, entering text, and pressing dialog box buttons. The following example is comprised of four types of dialog controls: an OK button, a Cancel button, three option buttons, and a group box.



Each control has its own set of underlying attributes. Each of the four types of controls above, and almost every other dialog control, have a height, width, horizontal location, vertical location, and text attribute. Additionally, most controls also have a value attribute. The value is a variable that is assigned string and number values that reflect the state of the dialog box when it closes. These variables can then be used in a script to initiate actions dependent on their values.

In some cases controls don't return a value to a script because their only function is to organize the dialog box controls or to provide the user with information. In the above example, the group box is the only control that doesn't return a value to a running script but provides a logical grouping for the option buttons.

Corel SCRIPT supports most sophisticated Windows dialog controls and the following dialog box provides an example of every dialog control available in Corel SCRIPT. See Dialog controls summary for more information.

{button ,AL(`Dialog_controls_summary;Returning_dialog_settings_and_choices;Corel_SCRIPT_Dialog_Editor;Sizing_and_placing_dialogs_and_controls;Default_control_sizes_and_labels;',0,"Defaultoverview",)} Related Topics

Dialog controls summary

The following table lists all the dialog controls available in Corel SCRIPT.

Control	Syntax (static reference)	Static dialog box return values	Example (click pop-up)
Bitmap button	BITMAPBUTTON	ID value (cannot equal 1 or 2) and closes the dialog	
Cancel button	CANCELBUTTON	2 and closes the dialog	
Check box	CHECKBOX	0 if disabled 1 if enabled 2 if mixed state	
Combo box	COMBOBOX	string value	
Drop-down combo box	DDCOMBOBOX	string value	
Drop-down list box	DDLISTBOX	integer value corresponding to an array	
Group box	GROUPBOX	doesn't return a value	
Help button	HELPBUTTON	opens a help file at a specified topic (does not close dialog box).	
Horizontal Slider	HSLIDER	a value from 0 to 100, inclusive	
Image List Box	IMAGELISTBOX	string value	
Image	IMAGE	doesn't return a value	
List box	LISTBOX	integer value corresponding to an array	
OK button	OKBUTTON	1 and closes the dialog	
Option button	OPTIONBUTTON	value returns to OPTIONGROUP not the dialog	
Option group	OPTIONGROUP	integer value corresponding to the option button selected	
Progress Indicator	PROGRESS	doesn't return a value	
Push button	PUSHBUTTON	ID value (cannot equal 1 or 2) and closes the dialog	
Spin control	SPINCONTROL	numeric value	
Status control	STATUS	doesn't return a value	
Text box	TEXTBOX	string value	
Text	TEXT	doesn't return a value	
Vertical Slider	VSLIDER	a value from 0 to 100, inclusive	

{button ,AL(`Corel_SCRIPT_dialog_controls;Returning_dialog_settings_and_choices;Working_with_dialog_controls;Default_control_sizes_and_labels;Corel_SCRIPT_Dialog_Editor;',0,"Defaultoverview",)} Related Topics

Static and dynamic dialog boxes

Corel SCRIPT dialog boxes come in two types: static and dynamic.

Static dialog boxes

A static dialog box, as its name indicates, doesn't change; that is, users can manipulate the dialog controls and then click a push button to close the dialog controls. Except for the control settings, static dialog boxes cannot be altered in appearance.

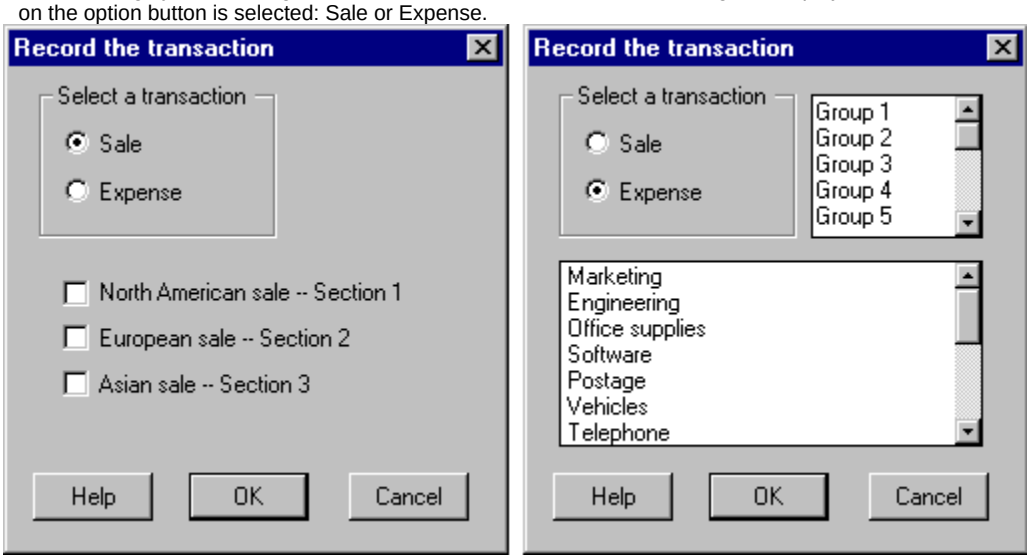
See Returning static dialog box settings to a script for more information about using static dialog boxes.

Dynamic dialog boxes

Unlike static dialog boxes, dynamic dialog boxes can change their display contents based on user action such as clicking a push button or selecting a list item. Most Windows applications use dynamic dialog boxes. For example, click Tools, Options in the Corel SCRIPT Editor to open the Options dialog box: clicking on a tab changes the dialog box appearance.

Dynamic dialog boxes are created the same way as static dialog boxes using the Corel SCRIPT Editor and dialog windows to create dialog definition statements. However, the definition statements in static and dynamic dialog boxes are not used the same way. Dynamic dialog boxes must respond each time the dialog box receives an action or an event occurs, while static dialog boxes do not have to respond immediately.

The following dynamic dialog box was created in Corel SCRIPT. This dialog box displays and hides different controls depending on the option button is selected: Sale or Expense.



Note

- The **BEGIN DIALOG** and **END DIALOG** statements on their own cannot display a static or dynamic dialog box and hold return values during a Corel SCRIPT script execution. Use the **DIALOG** statement to display the dialog box.
- Corel SCRIPT dialog boxes are modal; that is, a running script cannot continue until a dialog box is closed. However, a dynamic dialog box can open other dialog boxes, therefore, more than one dynamic dialog box can be open on the desktop.

{button ,AL(`Returning_dialog_settings_and_choices;Dynamic dialog box syntax;Dynamic Dialog Event Handler subroutine;;;',0,"Defaultoverview",)} Related Topics

Returning static dialog box settings to a script

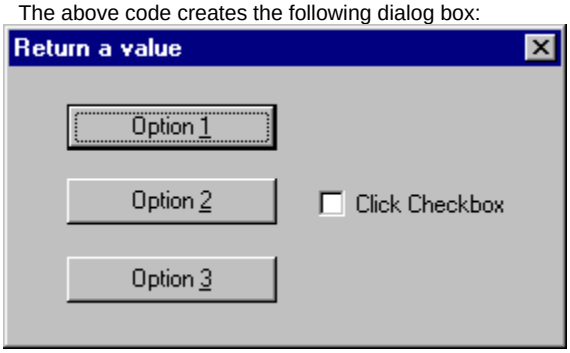
Static dialog boxes are a simple way for users to provide information that can be used with other script operations. The information comes from the dialog control settings when the dialog box is closed. Each dialog control has a value associated with it that corresponds to its setting.

Example

The following example shows how a static dialog box can return a value to a script.

```
BEGIN DIALOG Dialog1 16, 1, 184, 84, "Return a value"
    PUSHBUTTON  19, 11, 70, 14, "Option &1"
    PUSHBUTTON  19, 34, 70, 14, "Option &2"
    PUSHBUTTON  19, 58, 70, 14, "&Option &3"
    CHECKBOX  103, 37, 66, 10, "Click Checkbox", Y%
END DIALOG
'
' the next line defines the dialog return variable called "X"
X = DIALOG(Dialog1)    'this statement also displays the dialog box
```

The above code creates the following dialog box:




```
IF X = 3 THEN
    MESSAGE "Option 1 chosen"
ELSEIF X = 4 THEN
    MESSAGE "Option 2 chosen"
ELSE
    MESSAGE "Option 3 chosen"
ENDIF

IF Y = 0 THEN
    MESSAGE "Check box disabled"
ELSEIF Y = 1 THEN
    MESSAGE "Check box enabled"
ELSE
    MESSAGE "Check box in gray state"
ENDIF
```

The above script code opens a message box dependent on the dialog button pressed. The following message box is displayed if the "Option 2" was selected:

The variable **x** holds the dialog return value and is used in conditional statements to initiate further actions. The **Y** variable holds a value corresponding to the check box state.

Note

- Each Corel SCRIPT static dialog control that returns a value, does so in its own manner. For more information on how a dialog control returns a value, see its syntax reference and its respective example. See Dialog controls summary for a complete listing of Corel SCRIPT dialog controls.
- Corel SCRIPT dialog boxes are modal; that is, the running script cannot continue until the dialog box is closed.
- Neither the **BEGIN DIALOG** nor the **END DIALOG** statement on its own can display a dialog box; the **DIALOG** function displays a dialog box after it has been defined with the **BEGIN DIALOG** and **END DIALOG** statements.

{button ,AL(`Dialog;Static and dynamic dialog boxes;Corel_SCRIPT_dialog_controls;Dialog_controls_summary;Transferring_and_saving_Corel_SCRIPT_statements;Viewing_a_dialog_boxs_Corel_SCRIPT_statements;Working_with_dialog_controls;',0,"Defaultoverview",)} **Related Topics**

Changing focus in dialog boxes

Within a dialog box, only one control can have focus. Focus means that the control is active and can accept user input such as text or option selections. A dialog box's initial focus goes to the first control defined after the **BEGIN DIALOG** statement in a dialog box definition.

To move from control to control in a dialog box, you can use the TAB key. As you move around, the dialog boxe's focus changes. A dialog box convention is to have the focus move from left to right and top to bottom as you TAB through the controls. The TAB order is based on the order in which the controls are defined in the dialog box definition statements. You should try to place your control statements between the **BEGIN DIALOG** and **END DIALOG** statements in the order in which you want to TAB through a dialog.

Shortcut keys

An alternative to using the TAB key to move around a dialog box and change focus is to use shortcut keys. By placing an ampersand (&) before a character in a dialog box control's label you create a shortcut key. For example, in the following dialog box, pressing ALT+D changes the dialog focus to the first push button.

Dialog box controls that don't have labels, such as text boxes or list boxes, can also take on shortcut keys if you create an association with a text control. To associate a text control with a dialog control, the text control statement in the Corel SCRIPT script must precede the unlabelled control statement. Association is not based on a control's dialog location.

The following example shows two text boxes with defined shortcut keys in a static dialog box:


```
BEGIN DIALOG Dialog1 55, 10, 97, 89, "Shortcut key example"
    TEXT 5, 4, 80, 8, "&Enter text here:"
    TEXTBOX 5, 13, 80, 12, text1$
    TEXT 5, 31, 80, 8, "&Here too:"
    TEXTBOX 5, 40, 80, 12, text2$
    OKBUTTON 5, 64, 80, 14
END DIALOG
```

Note

- The initial focus of a dialog box should be a text box control, so that a user can start typing immediately once the dialog opens.
- Pressing ENTER is a shortcut for clicking the OK button and the Close Dialog button ( ) is a shortcut for the clicking the Cancel button.
- To use a ampersand (&) in a dialog control's text label, it must be preceded by another ampersand, for example, "Cats && Dogs"

{button ,AL(`csde;;;;;',0,"Defaultoverview",)} **Related Topics**

Corel SCRIPT dialog box tips

Editing dialog box tips

- You can insert multiple controls of the same type by holding down the CTRL key as you select a control from the Control menu or from the Control ribbon bar.
- The dialog Grid and the Snap to Grid options can help you align and distribute your dialog controls.
- List boxes and drop-down list boxes return integers that correspond to a selection from the array displayed in the control. The first array item returns 0, the second returns 1 and so on.
- You can select more than one dialog control by holding down the SHIFT key while selecting, or by clicking the mouse and dragging across the dialog box in a dialog window.

General dialog box tips

- In some cases, changing the screen resolution will change a dialog box's appearance. If your dialog boxes will be used on a variety of screens at different resolution settings, you should test the dialog boxes at each setting to ensure they will be properly displayed.
- Instead of a custom dialog box, consider using a predefined Corel SCRIPT dialog box. Click for more information.
- Neither the BEGIN DIALOG nor the END DIALOG statement on its own can display a dialog box. The **DIALOG** function displays a dialog box after it has been defined with the BEGIN DIALOG and END DIALOG statements.
- Pressing the Close Dialog button ( ) is the same as pressing the Cancel button; both return 2 to the dialog box.
- Use push buttons and bitmap buttons to open other dialog boxes. For example, within a script consider using a **CASE** statement to open another dialog box after a user has pressed a push button.
- Pressing ENTER is a shortcut for clicking the OK button, and pressing ESC or the Close Dialog button ( ) is a shortcut for clicking the Cancel button.
- Corel SCRIPT dialog boxes are modal; that is, the running script cannot continue until the dialog box is closed. However, a dynamic dialog box can be used to open another dialog box.

{button ,AL(`Dialog_box_conventions;Corel_SCRIPT_and_dialog_boxes;Corel_SCRIPT_Dialog_Editor;;;',0,"Defaultoverview",)} Related Topics

Dialog box conventions

To make dialog boxes easier to understand at a quick glance, consider the following conventions when creating them:

- Dialog boxes are read left to right and top to bottom. The dialog controls should be arranged in a way that allows a user to easily read the controls. The tab order should also follow the left-to-right and top-to-bottom convention.
- The initial focus of a dialog box should be a text box control so that a user can start typing immediately once the dialog box opens.
- Use sentence formatting for dialog text titles; that is, the initial character of the sentence should be in uppercase. The sentence should end with a colon, not a period, for example, **Choose a file:**.
- Push button text should only have the initial character of each word in uppercase characters. For example, **Close File** not **CLOSE FILE**. If a push button option opens another dialog, the push button text should be followed by an ellipsis (...), for example, **Printer Options...** .
- Group and position option buttons and check boxes vertically. Both option buttons and check boxes should be grouped in a logical manner. Avoid grouping option buttons and check boxes together.
- Consider using a list box of some type when you are providing more than 6 choices.
- Provide a shortcut key for every dialog control. For controls that don't use labels, such as text boxes and list boxes, create a shortcut key association with a text control.
- Static text should be aligned to the control it is identifying.
- Each dialog box should include both an OK and a Cancel button. The two buttons should be aligned horizontally in the bottom-right of the dialog box, or aligned vertically in the top-right of a dialog box.

Sizing, aligning, and distributing dialog controls

- Dialog controls of the same type should be the same size, especially in the case of push buttons, check boxes, option buttons, and list boxes.
- Use a margin of 4 dialog units from the inside of the dialog box borders.
- Separate groups of controls by 6 units both horizontally and vertically. Controls within a group can be separated by 2 units, except for option buttons and check boxes which do not need any separation.
- Use the alignment and distribution tools to create a symmetrical look for dialog controls.

{button ,AL(`Corel_SCRIPT_Dialog_Tips;Corel_SCRIPT_and_dialog_boxes;Corel_SCRIPT_Dialog_Editor;Sizing_and_placing_dialogs_and_controls;Default_control_sizes_and_labels;',0,"Defaultoverview",)} Related Topics

Predefined dialog boxes in Corel SCRIPT

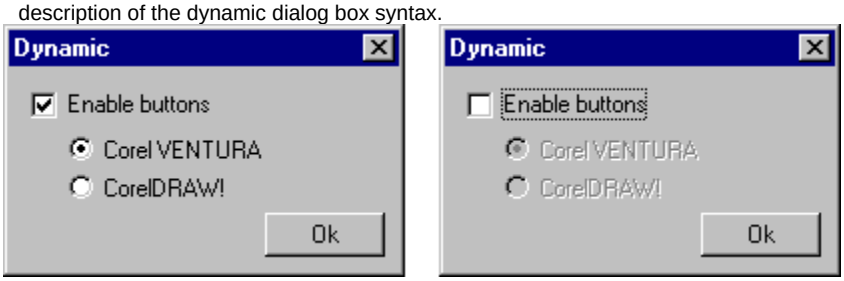
Corel SCRIPT provides predefined dialog boxes for your scripts. Most of these dialog boxes perform only a single function, but are easy to use and can save you the time it takes to program a custom dialog box . See the following Corel SCRIPT statements for more information about using predefined dialog boxes:

- **GETCOLOR**
- **GETFILEBOX**
- **GETFOLDER**
- **GETFONT**
- **INPUTBOX**
- **MESSAGEBOX**
- **MESSSAGE**

{button ,AL(`Corel_SCRIPT_Dialog_Tips;Corel_SCRIPT_and_dialog_boxes;;;;',0,"Defaultoverview",)} Related Topics

Dynamic dialog box syntax

Dynamic and static dialog boxes are both created using Corel SCRIPT dialog definition statements. However, the syntax of each dynamic dialog definition statement differs slightly from its static counterpart. The following script creates a dynamic dialog box which enables and disables option buttons depending on the state of the checkbox. Click the green hot spots in the script for a description of the dynamic dialog box syntax.




```
BEGIN DIALOG OBJECT Test 128, 88, 130, 62, "Dynamic", SUB Example1
    CHECKBOX  7, 5, 74, 13, .CheckBox1, "Enable buttons"
    OPTIONGROUP .OptionGroup1
        OPTIONBUTTON 19, 20, 68, 10, .OptionButton1, "Corel VENTURA"
        OPTIONBUTTON 19, 32, 68, 10, .OptionButton2, "CorelDRAW!"
    OKBUTTON  85, 44, 40, 14, .OK1
END DIALOG

Test.CheckBox1.SETVALUE(1)
DIALOG Test

'The following section is a Dialog Event Handler subroutine
SUB Example1(BYVAL Control%, BYVAL Event%)

DIM ReturnValue%
IF Event = 0 THEN
    Test.CheckBox1.SETTHREESTATE FALSE
ENDIF

IF Event = 2 THEN
    ReturnValue = Test.CheckBox1.GETVALUE()
    Test.OptionButton1.ENABLE ReturnValue
    Test.OptionButton2.ENABLE ReturnValue
ENDIF

END SUB
```

{button ,AL(`Static and dynamic dialog boxes;Dynamic Dialog Event Handler subroutine;Dialog Event Handler subroutine syntax;Using dynamic dialog box functions;Dynamic_dialog_functions_listing;',0,"Defaultoverview",)}
Related Topics

Dynamic Dialog Event Handler subroutine

The biggest difference between a dynamic and static dialog box is that a dynamic dialog box uses a Dialog Event Handler subroutine. This user-defined subroutine is specified in the **BEGIN DIALOG** statement. The Dialog Event Handler subroutine sends script instructions to a dynamic dialog box when an event or changes occur in a dialog box. For example, clicking a checkbox is a dialog event that can be used to trigger a change in a dialog box such as enabling or hiding a control.

The Dialog Event Handler subroutine is first called when a dynamic dialog box is initialized (opened and displayed with the **DIALOG** statement). After it is initialized, the subroutine is called continuously until the dialog box is closed.

In the following script example, both statements begin a dialog box definition. The first statement is used with a static dialog box and the second statement is used with a dynamic dialog box. The second statement specifies **Example1** as the Dialog Event Handler subroutine. Note that the dynamic dialog box syntax also uses the OBJECT keyword.

```
BEGIN DIALOG Test 128, 88, 130, 62, "Static"    '<-- Static dialog box
BEGIN DIALOG OBJECT Test 128, 88, 130, 62, "Dynamic", SUB Example1 '<-- Dynamic dialog box
```

See Dialog Event Handler subroutine syntax for more information.

{button ,AL(`Script_procedures ;SUB_end_sub;Static and dynamic dialog boxes;all_dialog_events;;;;;',0,"Defaultoverview",)} Related Topics

Dialog Event Handler subroutine syntax

SUB Subname(BYVAL ControlID%, BYVAL EventCode%)

The above syntax must be used to specify a Dialog Event Handler subroutine.

Parameter	Description
Subname	Specifies the name of Dialog Event Handler subroutine. The name of the subroutine must correspond to the subroutine specified in dynamic dialog box's **BEGIN DIALOG** statement.

ControlID	Specifies the variable that is passed a value corresponding to the dialog control that is generating a dialog event. The values correspond to the control's position in the dialog box's definition statements. The first listed control is identified as 1, the second listed control is identified as 2, and so on. **NOTE:** Although, OPTIONGROUP is not a control, it still uses a position value.
	The following example illustrates **ControlID** numbering:
	`BEGIN DIALOG...`
	`  CHECKBOX 7, 5, 74, 13, .CheckBox1, "Enable buttons"`
	`  OPTIONGROUP OptionGroup1%`
	`  OPTIONBUTTON 19, 20, 68, 10, .OptionButton1, "Corel VENTURA"`
	`  OPTIONBUTTON 19, 32, 68, 10, .OptionButton2, "CorelDRAW!"`
	`  OKBUTTON  85, 44, 40, 14, .OK1`
	`END DIALOG`
	The check box is identified as 1, the option group is identified as 2, the first option button is identified as 3, and so on.

Event	Specifies the variable that is passed a value corresponding to the dialog event that occurs in the dialog box. The following events occur in a dialog box: 0 Dialog Initialization 1 Change in content 2 Clicking a control 3 Double-clicking a list box item 4 Change in focus 5 Time out 6 Resizing dialog box

Example

The following example shows the format of a typical Dialog Event Handler subroutine. This example doesn't show the instructions sent to dynamic dialog box, but shows how to use the conditional **IF...THEN...ENDIF** statements to set up a Dialog Event Handler subroutine. For more information about the instructions sent to dynamic dialog boxes, see Using dynamic dialog box functions. This example uses the same control examples used in the **ControlID** parameter, above. Click the green hot spots in the script for a description of the dynamic dialog box syntax.

```
SUB Example1(BYVAL ControlID%, BYVAL Event%)

IF Event = 0 THEN
    REM send these instructions to the dialog box
ENDIF

IF Event = 2 THEN
    REM send these instructions to the dialog box
ENDIF

IF ControlID = 1 THEN
    REM send these instructions to the dialog box
ENDIF

IF Event = 2 THEN
    IF ControlID = 1 THEN
        REM send these instructions to the dialog box
    ENDIF
    IF ControlID = 3 THEN
```

```
        REM send these instructions to the dialog box
    ENDIF
ENDIF

END SUB
```

Note

- Both Dialog Event Handler subroutine parameters must be passed by value (BYVAL).

{button ,AL(`all_dialog_events;;;;;',0,"Defaultoverview",)} **Related Topics**

Using dynamic dialog box functions

To set, change, or return attributes and settings in a dynamic dialog box, special dynamic dialog box functions must be used. The following section compares using controls in static dialog boxes to using controls in dynamic dialog boxes.

In static dialog boxes, use the control statement to set and to return a dialog control's attributes. For example, the following line from a script sets a the attributes of a check box:

```
LISTBOX 4, 14, 90, 50, ArrayList, SelectNum
```

The four numeric parameters set list box positioning, the **ArrayList** variable sets the array to use in the list box, and the **SelectNum** variable sets the default selection and holds the return value that corresponds to the selected list box entry.

To set the same options in a dynamic dialog box, the following lines of script code are required:

```
LISTBOX 4, 14, 90, 50, .ListBox1                'Line 1
DialogName.ListBox1.SETARRAY ArrayList          'Line 2
DialogName.ListBox1.SETSELECT SelectNum         'Line 3
ReturnValue = DialogName.ListBox1.GETSELECT( )  'Line 4
```

- Line 1 is similar to the static statement; it sets the list box positioning but also creates an identifier (**ListBox1**) for the list box. This identifier is used to send messages to the list box. This line must be in the dialog box definition.
- Line 2 uses the **SETARRAY function** to set the array used in the list box. The function is preceded by the control's identifier (**ListBox1**) and the dialog box's identifier (**DialogName**). The identifiers and functions must be separated by periods. Initial settings for a dialog box and its controls are normally set in the dialog box's Dialog Event Handler subroutine in the Dialog Initialization event.
- Line 3 is similar to Line 2, except that it uses the **SETSELECT** function to set the default list box selection.
- Line 4 uses the **GETSELECT** function to return the selection in the list box to the **ReturnValue** variable. Normally, a function that returns a value appears after the dialog initialization statement (**DIALOG**), or within the Dialog Event Handler subroutine.

The above four Corel SCRIPT lines duplicate the functionality of a listbox in a static dialog box. However, you have more control over dynamic dialog boxes and their controls than their static counterparts. For example, you can also use the **ENABLE** function to disable a list box or the **SETSTYLE** function to hide the list box.

Each dialog control and dialog box uses a unique set of dialog box functions. To find the functions a dialog box or a control can use, see the dynamic dialog box and control syntax in the online help. The syntax overview for each dynamic dialog box function also indicates which dialog items they can be used with. See Dynamic dialog functions listing for a list of dynamic dialog box functions.

{button ,AL(`Dialog Event Handler subroutine syntax;Returning_dialog_settings_and_choices;Static and dynamic dialog boxes;Dynamic_dialog_functions_listing;;',0,"Defaultoverview",)} Related Topics

Dynamic dialog functions listing

The following functions can be used to set control values and settings in dynamic dialog boxes.

ADDITEM
CLOSEDIALOG
ENABLE
GETBITMAPHEIGHT
GETBITMAPWIDTH
GETHEIGHT
GETHELPINDEX
GETHELPPATH
GETID
GETIMAGE
GETINCREMENT
GETITEM
GETITEMCOUNT
GETLEFTPOSITION
GETMAXRANGE
GETMINRANGE
GETPRECISION
GETSELECT
GETSTYLE
GETTEXT
GETTICK
GETTIMER
GETTOPPOSITION
GETVALUE
GETWIDTH
MOVE
REMOVEITEM
RESET
SETARRAY
SETBITMAPOFFSET
SETDOUBLEMODE
SETHELPINDEX
SETHELPPATH
SETIMAGE
SETINCREMENT
SETMAXRANGE
SETMINRANGE
SETPRECISION
SETSELECT
SETSTYLE
SETTEXT
SETTHREESTATE
SETTICK
SETTIMER
SETVALUE
SETVISIBLE
STEP

{button ,AL(`Using dynamic dialog box functions;Dialog Event Handler subroutine syntax;Dynamic Dialog Event Handler subroutine;;;',0,"Defaultoverview",)} **Related Topics**

Dialog Event 0: Dialog Initialization

The Dialog Initialization event occurs just before a dynamic dialog box is initialized. Initializing a dialog box means that it is being opened and displayed with the **DIALOG** statement. A dynamic dialog box can only be initialized once ; that is, a dialog event equal to 0 can only be passed to the Dialog Event Handler subroutine one time.

The initialization event is the time to set the opening attributes for the dialog box and controls within it. By default, all dynamic dialog controls are enabled and visible. If you wanted to open a dialog box with some controls disabled or hidden, use Dialog Event 0 in the Dialog Event Handler subroutine. Other typical uses for the Dialog Initialization event include setting default text in text boxes, specifying arrays for list boxes, and check box and spin control attributes.

Once the dialog box is initialized, Corel SCRIPT displays it based on the specifications in the dialog box's definition and the Dialog Event Handler's Dialog Initialization condition.

Note

- Since a control is not used in generating the Dialog Initialization event, the **ControlID** parameter in the Dialog Event Handler subroutine does not have a valid return value.
- You can also set the opening attributes for a dialog box by placing dynamic dialog box functions before the **DIALOG** statement in a script.

{button ,AL(`all_dialog_events;enable;getstyle;setstyle;;',0,"Defaultoverview",)} Related Topics

Dialog Event 1: Change in content

The Change in content event occurs whenever:

- text is typed into a text box control
- text is typed into the text box portion of a combination box control
- arrow keys on the keyboard are used to move the slider indicator on a horizontal or vertical slider
- a spin control value setting changes by either typing in the text box portion or using the arrow keys

Note

- In addition to the Change in content event, you should consider using the Change in focus event when using text box controls.
- When the **Event** parameter in the Dialog Event Handler subroutine is 1 (Change in content), the **ControlID** parameter corresponds to the dialog control that is changed.

{button ,AL(`all_dialog_events;;;;;',0,"Defaultoverview",)} Related Topics

Dialog Event 2: Change in control

The Change in control event is the most common type of event in a dynamic dialog box and generally occurs whenever a push button is clicked or a selection is made in a check box, option button, or any type of list box. This event also occurs whenever the mouse is used to drag the slider indicator on a horizontal or vertical slider.

Note

- When the **Event** parameter in the Dialog Event Handler subroutine is 2 (Change in control), the **ControlID** parameter corresponds to the dialog control that is changed.

{button ,AL(`all_dialog_events;;;;;',0,"Defaultoverview",)} **Related Topics**

Dialog Event 3: Double-clicking a list box item

The Double-clicking a list box item event occurs whenever a user double-clicks an item in one of the five list boxes included in Corel SCRIPT:

- β λιστ βοξLISTBOX_dyn
- drop-down list boxDDLISTBOX_dyn
- combo boxCOMBOBOX_dyn
- drop-down combo boxDDCOMBOBOX_dyn
- image list boxIMAGELISTBOX_dyn

This event is often used as a shortcut for closing a dialog box or to update a dynamic dialog box's contents.

Note

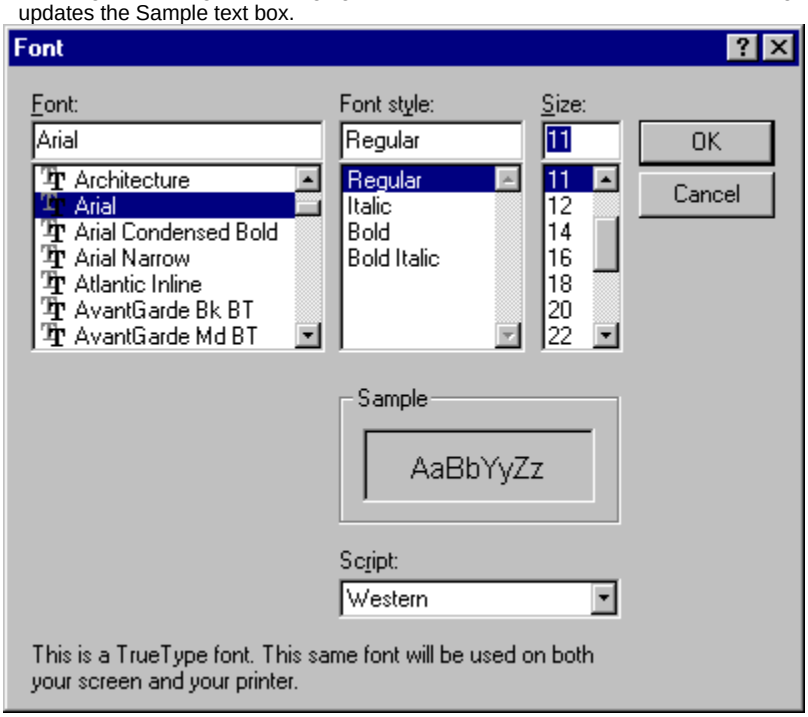
- When the **Event** parameter in the Dialog Event Handler subroutine is equal to 3 (Double-clicking a list box item), the **ControlID** parameter corresponds to the dialog control that is double-clicked.

{button ,AL(`Script_programming_errors;all_dialog_events;Corel_SCRIPT_Editor_Debugging_Features;message;;',0,"Defaultoverview",)} Related Topics

Dialog Event 4: Change in focus

The Change in focus event occurs whenever a control in a dynamic dialog box loses focus. Focus means that the control is active and can accept user input such as text or option selections. Only one control can have focus. A dialog box's initial focus goes to the first control defined after the **BEGIN DIALOG** statement in a dialog box definition. Controls lose focus whenever another control dialog control is selected or when you use the TAB or arrow key to move focus. The TAB order is based on the order in which the controls are defined in the dialog box definition statements.

The Change in focus event is most often used to update dialog control settings after a user has typed into a text box control and has changed dialog box focus to another control. Many common Windows dialog boxes exhibit this behavior. For example, in the following Font dialog box, changing the value in the Size text box and then moving the dialog focus to another dialog control updates the Sample text box.



Note

- When the **Event** parameter in the Dialog Event Handler subroutine is equal to 4 (Change in focus), the **ControlID** parameter corresponds to the dialog control that gains focus.
- You should try to place your control statements between the BEGIN DIALOG and END DIALOG statements in the order in which you want to TAB through a dialog.
- You should be careful debugging a script using the Change in focus event and a message box. It is easy to create a endless loop that continually closes and displays a message box.

{button ,AL(`Script_programming_errors;all_dialog_events;Corel_SCRIPT_Editor_Debugging_Features;message;;',0,"Defaultoverview",)} Related Topics

Dialog Event 5: Timer

The Timer event occurs when the dialog timer runs down to or equals 0. By default, dynamic dialog boxes do not use a timer; you must use the **SETTIMER** function to set a dialog box's timer value. The following line sets the timer value for **MyDialog** dynamic dialog box to 3 seconds (or 3000 milliseconds):

```
MyDialog.SETTIMER(3000)
```

Using this setting, the **MyDialog** dialog box calls its Dialog Event Handler subroutine every 3 seconds looking for an event condition equal to 4. The following example from a Dialog Event Handler subroutine displays a message box each time the Timer event is called:

```
IF Event = 5 THEN
    MESSAGE "Another 3 seconds of your life has passed"
ENDIF
```

Once the subroutine call has terminated, the timer is reset to its last **SETTIMER** value and starts to count down again.

You can use this event to automatically initiate an action such as closing a dynamic dialog box after a specified amount of time.

Note

- The **SETTIMER** function should be placed before the **DIALOG** function in a script; it should not normally be placed in the Dialog Event Handler subroutine, except to change it's value.
- Because of system speed limitations, setting the timer value to a low number may not give desired results. For example, setting the timer to 300 milliseconds may actually result in the timer using more than 0.3 seconds.

{button ,AL(`closedialog;all_dialog_events;;;;;',0,"Defaultoverview",)} Related Topics

Dialog Event 6: Resizing dialog box

The Resizing dialog box event occurs when a custom dynamic dialog box is resized. A dialog box can only be resized if the **SETSTYLE** function is set to 2048. You can use this event to change dialog box control settings when you allow users to resize a dialog box. For example, if you allow users to resize a dialog box that uses an image control, you can set the image control so that it takes up a specified proportion of the dialog box size. Click the Example button above for more information.

{button ,AL(`closedialog;all_dialog_events;;;;;',0,"Defaultoverview",)} **Related Topics**

To print my own Corel SCRIPT manual

1. From a Corel SCRIPT Help window, click Contents or Help Topics button.
2. Click the Contents tab.
3. Click the book or sub-book you want to print and then click the Print button.
4. Check your printer settings, then click OK.

Note

Because Help prints one topic at a time and prints one topic per page, printing may take a lot of time and a lot of paper.

To start Corel SCRIPT Editor using the Windows 95/NT Run command

1. On the Windows desktop, click Start, Run.
2. In the Open edit box, type the Editor's folder location and SCEDIT.
   For example, C:\COREL\GRAPHICS\PROGRAMS\SCEDIT
   Corel applications typically reside in the COREL\GRAPHICS\PROGRAMS folder.

Note

- If you forget the location of the Corel SCRIPT Editor, click Browse.
- After the program is started, the Corel SCRIPT Editor button is displayed in the Windows taskbar.

{button ,AL(`a_start_de;;;;;',0,"Defaultoverview",)} **Related Topics**

To start the Corel SCRIPT Editor from a Corel application

From your Corel application:

- From CorelDRAW 6/7/8: click Tools, Scripts, Corel SCRIPT Editor.
- From CorelDRAW 9, click Tools, Corel SCRIPT, Corel SCRIPT Editor
- From Corel PHOTO-PAINT 6/7/8: click Tools, Scripts, Corel SCRIPT Editor.
- From Corel PHOTO-PAINT 9, click Tools, Corel SCRIPT, Corel SCRIPT Editor.
- From Corel VENTURA 7: click Tools, Scripts, Corel SCRIPT Editor.
- From Corel VENTURA 8: click Tools, Corel SCRIPT, Corel SCRIPT Editor.

{button ,AL(`a_start_de;;;;;',0,"Defaultoverview",)} **Related Topics**

To run a Corel SCRIPT script from a Corel application

1. Click Tools:
   - For CorelDRAW 6/7/8: click Scripts, Run Script.
   - For CorelDRAW 9, click Corel SCRIPT, Run Script.
   - For Corel PHOTO-PAINT 6/7: click Scripts, Run.
   - For Corel PHOTO-PAINT 8: click Scripts, Scripts.
   - For Corel PHOTO-PAINT 9, click Corel SCRIPT, Run Script.
   - For Corel VENTURA 7: click Tools, Scripts, Run / Manage Script.
   - For Corel VENTURA 8: click Tools, Corel SCRIPT, Run / Manage Script.
2. If the Corel SCRIPT script is not in the default folder, chose the drive and folder where it is stored.
3. Double-click the Corel SCRIPT script you want to run. From Corel VENTURA, select the script you want to run and click Run. From Corel PHOTO-PAINT 8, select the script you want to run and click the play button.

Notes

- You can use wild cards (* and ?) if you're not sure of the name of the file you want to run. For example, typing **script*.csc** in the File Name box and clicking OK lists all CSC files in the selected folder beginning with **script**. Typing **sc?.csc** in the File Name box and clicking OK lists all CSC files in the selected folder that begin with **sc** and are followed by only one more character.
- To run a script from your Corel application, the script must contain the **WITHOBJECT** construct.
- You can't undo executed script commands. If you're not sure whether a script is running to your specifications, then before the script is executed, save the documents in the Corel application that will receive the script instructions .
- You can terminate a script's execution by pressing CTRL+BREAK.

{button ,AL(`app_Corel_SCRIPT;script_files;ht_play_script_cse;;;',0,"Defaultoverview",)} Related Topics

To start the Corel SCRIPT Editor from the Windows 95 desktop

1. On the Windows desktop, click Start, Programs.
2. Point to the folder that contains the Corel SCRIPT Editor if it does not appear on the main Program menu.
   Corel applications normally reside in the COREL folder.
3. Click Corel SCRIPT Editor (scedit.exe).

Note

- After the program is started, the Corel SCRIPT Editor button is displayed in the Windows taskbar.

{button ,AL(`a_start_de;;;;;',0,"Defaultoverview",)} **Related Topics**

To start the Corel SCRIPT Editor from Windows NT

- From the Windows Program Manager, double-click the Corel SCRIPT Editor.

Note

- The Corel SCRIPT Editor icon normally resides in a Corel programs window group.

{button ,AL(`a_start_de;;;;;',0,"Defaultoverview",)} **Related Topics**

To start the Corel SCRIPT Editor with startup options

Startup options are commands that you can use when you start the Corel SCRIPT Editor.

1. On the Windows taskbar, click Start, Run.
2. Type **C:\COREL\GRAPHICS\PROGRAMS\SCEDIT.EXE** followed by a space, then type the startup options you want. The **C:\COREL\GRAPHICS\PROGRAMS** folder is based on a typical installation.

Startup options	Action executed
SCEDIT.EXE	Starts the Editor with a new script.
SCEDIT.EXE *file*	Starts the Editor and opens a previously saved script. *file* specifies the name and path of the script to open.
SCEDIT.EXE /p *file*	Prints a previously saved script to the default printer port. *file* specifies the name and path of the script to print.
SCEDIT.EXE /pt *file port*	Prints a previously saved script to a specified printer port. *file* specifies the name and path of the script to print. *port* specifies the printer port.
SCEDIT.EXE /r *file*	Executes a previously saved script or Corel SCRIPT Binary file. *file* specifies the name and path of the script or Binary to execute.
SCEDIT.EXE /n	Starts the Corel SCRIPT Editor without displaying the splash screen.
SCEDIT.EXE *file* /exe *name*	Creates a Corel SCRIPT Executable from a script. *file* specifies the name and path of the script to use. *name* specifies the name and, optionally, the path of the executable to create.
SCEDIT.EXE *file* /dll *name*	Creates a Corel SCRIPT dynamic link library (DLL) from a script. *file* specifies the name and path of the script to use. *name* specifies the name and, optionally, the path of the DLL to create.
SCEDIT.EXE *file* /cao *name*	Creates a Corel Add-on (.CAO) from a script. *file* specifies the name and path of the script to use. *name* specifies the name and, optionally, the path of the Add-on to create.
SCEDIT.EXE *file* /csb *name*	Creates a Corel SCRIPT Binary (.CSB) from a script. *file* specifies the name and path of the script to use. *name* specifies the name and, optionally, the path of the Binary to create.
SCEDIT /c *commandline*	Starts the Editor and *commandline* sets the command line.

Note

- When you use more than one start-up option, add a space between options.

{button ,AL(`Creating Corel SCRIPT Executables;GETCOMMANDLINE ;a_start_de;;;;;',0,"Defaultoverview",)} Related Topics

To create a new script with the Corel SCRIPT Editor

- Click File, New. An untitled document window opens.

Note

- You can have multiple document windows opened in the Corel SCRIPT Editor.

{button ,AL(`ht_file_menu_cse;;;;;',0,"Defaultoverview",)} Related Topics

To open a Corel SCRIPT script

1. Click File, Open.
2. If the Corel SCRIPT script is not in the default folder, chose the drive and folder where the Corel SCRIPT script is stored.
3. Double-click the Corel SCRIPT script you want to open.

Note

- You can use wild cards (* and ?) if you're not sure of the name of the file you want to open. For example, typing **script*.csc** in the File Name box and clicking OK lists all CSC files in the selected folder beginning with **script**. Typing **sc?.csc** in the File Name box and clicking OK lists all CSC files in the selected folder that begin with **sc** and are followed by only one more character.

{button ,AL(`ht_file_menu_cse;;;;;',0,"Defaultoverview",)} **Related Topics**

To close a Corel SCRIPT script

- Click File, Close.

Notes

- If your changes have not been saved, a confirmation message appears.
- To use this command, the script window you want to close must be active.
- If you close a script window that has launched a still open dialog window, the dialog window also closes.

{button ,AL(`ht_file_menu_cse;;;;;',0,"Defaultoverview",)} **Related Topics**

To save a Corel SCRIPT script

- Click File, Save.

Notes

- If you're saving a new Corel SCRIPT script, type a name in the File name box.
- If you issue this command with a dialog window active, Corel SCRIPT Editor saves the script that launched the dialog window. See Script windows and the Corel SCRIPT Editor for more information.
- To save a Corel SCRIPT script with a new name, click File, Save As and type a new name in the File Name box.
- To save all open scripts, click File, Save All.

{button ,AL(`ht_file_menu_cse;script_files;;;;',0,"Defaultoverview",)} Related Topics

To print a Corel SCRIPT script

- Click File, Print.

Notes

- You can only issue this command if a script window is active.
- Click the Setup button from Printer dialog or click File, Print Setup to set the paper size and orientation as specified by the active printer.

Tips

- If your script has lines longer than 80 characters, click File, Print Setup to change page orientation to landscape. The Corel SCRIPT Editor does not automatically wrap long lines when printing.
- You can't print a dialog box from the Corel SCRIPT Editor directly, but you can work around this: First, press PRINT SCREEN on your keyboard to capture the screen contents to the Clipboard or ALT+PRINT SCREEN to capture the active window's contents. Then open a graphics program such as Corel PHOTO-PAINT or the Windows Paintbrush and paste the Clipboard contents. From either one of these applications you can edit your screen capture and print.

{button ,AL(`ht_file_menu_cse;;;;;',0,"Defaultoverview",)} Related Topics

To create a Corel SCRIPT Executable

1. Click File, Make EXE.
2. Type a name in the File name box.
3. Click Save.

Notes

- You can edit and change a script after it has been used to compile a Corel SCRIPT Executable without affecting the Executable.
- Executables require that the Corel SCRIPT run-time interpreter be installed on the user's system when run. The run-time interpreter is a dynamic link library named SCINT**xx**.DLL where **xx** indicates the Corel SCRIPT major version number ➪ for example SCINT70.DLL for Corel SCRIPT version 7.0. Click ➪ for a list of locations a Corel SCRIPT Executable searches for the run-time interpreter.

{button ,AL(`cs_exe_dll;ht_exe_dll;GETVERSION;;;',0,"Defaultoverview",)} Related Topics

To run a Corel SCRIPT Executable

There are several ways to run a Corel SCRIPT Executable in Windows 95 or NT. Here are a few:

- Right-click the Executable in the Windows Explorer and choose Open.
- On the Windows 95/NT 4.0 desktop, click Start, Run. Enter the Executable name in the Open text box and click OK.
- In Windows NT 3.51, select the Executable with the File Manager and click File, Run.

Notes

- Though you can't assign a Corel SCRIPT Executable directly to a toolbar in a Corel application, you can assign it to the Corel Application Launcher toolbar button. Click ➪ for more information.
- Corel SCRIPT Executables require that the Corel SCRIPT run-time interpreter be installed on the user's system when run. The run-time interpreter is a dynamic link library named SCINT**xx**.DLL where **xx** indicates the Corel SCRIPT major version number ➪ for example SCINT70.DLL for Corel SCRIPT version 7.0 . Click ➪ for a list of locations where a Corel SCRIPT Executable searches for the run-time interpreter.

{button ,AL(`cs_exe_dll;ht_exe_dll;GETVERSION;;;',0,"Defaultoverview",)} **Related Topics**

To create a dynamic link library

1. Click File, Make DLL.
2. Type a name in the File name box.
3. Click Save.

Notes

- DLLs created with Corel SCRIPT contain only one exportable function. This function is always named **RUN**. The **RUN** function executes the script commands in the script that was used to create the DLL.
- DLLs, created with Corel SCRIPT, require that the Corel SCRIPT run-time interpreter be installed on the user's system when run. The run-time interpreter is a dynamic link library named SCINT**xx**.DLL, where **xx** indicates the Corel SCRIPT major version number ⇨ for example, SCINT70.DLL for Corel SCRIPT version 7.0. Click ⇨ for a list of locations where a DLL created with Corel SCRIPT searches for the run-time interpreter.

{button ,AL(`addresbmp;cs_exe_dll;ht_exe_dll;;;;',0,"Defaultoverview",)} Related Topics

To create a Corel Add-on using Corel SCRIPT

1. Click File, Make CAO.
2. Type a name in the File name box.
3. Click Save.

Notes

- You can also write Corel Add-ons in a programming environment you are most familiar with such as Pascal, FORTRAN, C, or C++. See Developing Corel Add-ons for VENTURA for information about writing your own Add-ons.
- Corel Add-ons, created with Corel SCRIPT, require that the Corel SCRIPT run-time interpreter be installed on the user's system when run. The run-time interpreter is a dynamic link library named SCINT**xx**.DLL, where **xx** indicates the Corel SCRIPT major version number ➪ for example, SCINT70.DLL for Corel SCRIPT version 7.0. Click ➪ for a list of locations where a Corel Add-on created with Corel SCRIPT searches for the run-time interpreter.

{button ,AL(`addresbmp;cs_exe_dll;ht_exe_dll;;;;',0,"Defaultoverview",)} Related Topics

To create a Corel SCRIPT Binary

1. Click File, Make CSB.
2. Type a name in the File name box.
3. Click Save.

Notes

- Corel SCRIPT Binary files require that the Corel SCRIPT run-time interpreter be installed on the user's system when executed. The run-time interpreter is a dynamic link library named SCINT**xx**.DLL, where **xx** indicates the Corel SCRIPT major version number ⇨ for example, SCINT70.DLL for Corel SCRIPT version 7.0. Click ⇨ for a list of locations where a Corel SCRIPT Binary file created with Corel SCRIPT searches for the run-time interpreter.
- You can run a Corel SCRIPT Binary file by double-clicking its icon in the Windows Explorer. Click ⇨ for information about running Corel SCRIPT Binaries.
- Like scripts, Binaries can be assigned to shortcut keys, the menu system, or toolbar buttons. Click ⇨ for more information.

{button ,AL(`addresbmp;cs_exe_dll;ht_exe_dll;;;;',0,"Defaultoverview",)} Related Topics

To add or remove applications from the Corel Application Launcher

The Corel Application Launcher is a toolbar button  you can use to launch Corel applications, Corel SCRIPT Executables, and applications from other vendors. The list of applications on the Application Launcher is determined at the time of installation. You can add or delete other applications by editing the CORELAPP.INI which is located in the \COREL\CONFIG folder. Use a text editor such as the Windows NotePad or the Corel SCRIPT Editor to open the file, then locate the [Applications] section.

To add an application, type its name as you want it to appear on the Application Launcher drop-down menu. Then, type an equals sign (=) and the path to the folder where the application is located. Next, type the application's executable file name. Using Corel PHOTO-PAINT as an example, the entry should look like this:

[Applications]
Corel PHOTO-PAINT = C:\COREL\PROGRAMS\CORELPNT.EXE

To remove an application from the Application Launcher, delete its entry in the [Applications] section.

Notes

- If the Corel Application Launcher is not available on a toolbar in the Corel SCRIPT Editor, click Tools, Customize. Click the Toolbars tab, then click the Tools folder in the Command categories box. In the buttons section of the dialog box, click the Application Launcher icon to add it to the Toolbar.

{button ,AL(`cs_exe_dll;ht_exe_dll;cs_toolbar_cs;ht_ca_launching;;',0,"Defaultoverview",)} **Related Topics**

To launch applications from the Corel Application Launcher

1. Click the Corel Application Launcher toolbar button  .
2. Click the application you want to run.

Note

- If the Corel Application Launcher is not available on a toolbar in the Corel SCRIPT Editor, click Tools, Customize. Click the Toolbars tab, then click the Tools folder in the Command categories box. In the buttons section of the dialog box, click the Application Launcher icon  to add it to the Toolbar.

{button ,AL(`cs_exe_dll;ht_exe_dll;cs_toolbar_cs;ht_ca_launcher;;',0,"Defaultoverview",)} Related Topics

To close the Corel SCRIPT Editor

- Click File, Exit.

Note

- You are prompted to save any unsaved changes in any open documents.

{button ,AL(`ht_file_menu_cse;;;;;',0,"Defaultoverview",)} Related Topics

To undo editing operations

- Click Edit, Undo.

Notes

- You can only issue this command if a script window is active; that is, custom dialog editing in a dialog window can't be undone.
- You can't undo editing operations after the script has been saved.

{button ,AL(`ht_edit_cse;;;;;',0,"Defaultoverview",)} **Related Topics**

To restore changes reversed by the Undo command

- Click Edit, Redo.

Note

- You can only issue this command if a script window is active; that is, you can't use the Redo command when editing a custom dialog in dialog window.

{button ,AL(`ht_edit_cse;;;;;',0,"Defaultoverview",)} **Related Topics**

To copy text to another location

1. Select the text.
2. Click Edit, Copy.
3. Place the insertion point in the document window where you want paste the text.
4. Click Edit, Paste.

Note

- The selected text remains on the Clipboard until you cut or copy another selection to it from any Windows application.

{button ,AL(`ht_edit_cse;;;;;',0,"Defaultoverview",)} Related Topics

To cut text to move to another location

1. Select the text.
2. Click Edit, Cut.
3. Place the insertion point in the document window where you want paste the text.
4. Click Edit, Paste.

Note

- The selected text remains on the Clipboard until you cut or copy another selection to it from any Windows application.

{button ,AL(`ht_edit_cse;;;;;',0,"Defaultoverview",)} Related Topics

To delete text

1. Select the text you want to delete.
2. Click Edit, Delete.

   The selected text is not transferred to the Clipboard.

Notes

- You can also delete text without selecting it by pressing the BACKSPACE and DELETE key. The BACKSPACE key deletes text to the left of the insertion point and the DELETE key deletes text to the right of the insertion point.
- Instead of using Edit, Delete, you can delete text by clicking Edit, Cut which transfers controls from the script to the Clipboard.

{button ,AL(`ht_edit_cse;;;;;',0,"Defaultoverview",)} Related Topics

To place REM statements at the beginning of script lines

1. Place the insertion point in the line where you want to place a REM statement. If you want to place REM statements in a contiguous block of statements, select the statements.
2. Click Edit, Comment.

Notes

- Script lines that begin with **REM** statements are ignored during script execution. This feature can be useful during debugging sessions.
- Use the UnComment command to remove REM statements from selected lines in a script.

{button ,AL(`rem;ht_uncomment;ht_comment;;;',0,"Defaultoverview",)} Related Topics

To remove REM statements from the beginning of script lines

1. Place the insertion point in the line where you want to remove a REM statement. If you want to remove REM statements in a contiguous block of statements, select the statements.
2. Click Edit, UnComment.

Note

- Script lines that begin with **REM** statements are ignored during script execution. This feature can be useful during debugging sessions.

{button ,AL(`rem;ht_uncomment;ht_comment;;;',0,"Defaultoverview",)} **Related Topics**

To find text

1. Click in the document where you want to begin searching.
2. Click Search, Find.
3. Enter the text you want to find in the Find What box.
4. Click Find Next.

Tip

- To find and replace text, click Edit, Replace instead of Edit, Find.

{button ,AL(`ht_find_replace_cse;;;;;',0,"Defaultoverview",)} **Related Topics**

To find and replace text

1. Click in the document where you want to begin searching.
2. Click Search, Replace.
3. In the Find What box, enter the text you want to find.
4. In the Replace With box, enter the replacement text.
5. Click Replace All to replace all occurrences of the text to find.

Note

- To replace individual text occurrences, click Find Next, Replace instead of clicking Replace All.

{button ,AL(`ht_find_text_cse;;;;;',0,"Defaultoverview",)} **Related Topics**

To go to a line in the Corel SCRIPT Editor

1. Click Search, Go to Line.
2. In the Line number box, enter a line number.

{button ,AL(`next_prev;;;;;',0,"Defaultoverview",)} Related Topics

To go to the next error in a script

- Click Search, Next Error.

Notes

- Before running this command, the Compiler Output window must display at least one error message and the insertion point must be in the script window.
- The line where the insertion point is sent has the ⇨ symbol displayed in its left margin.

{button ,AL(`ht_debug;next_prev;;;;',0,"Defaultoverview",)} Related Topics

To go to the previous error in a script

- Click Search, Previous Error.

Notes

- Before running this command, the Compiler Output window must display at least one error message and the insertion point must be in the script window.
- The line where the insertion point is sent has the ⇨ symbol displayed in its left margin.

{button ,AL(`ht_debug;next_prev;;;;',0,"Defaultoverview",)} Related Topics

To view or hide Watch window

- Click View, Watch Window.

  A check mark beside the Watch Window menu command indicates the Watch window is displayed.

Note

- The Watch window can be resized by clicking on a border and dragging.

{button ,AL(`ht_statusbar;ht_watch;ht_compiler;Corel_SCRIPT_Editor_windows;;',0,"Defaultoverview",)} **Related Topics**

To view or hide Compiler Output window

- Click View, Compiler Output Window.

  A check mark beside the Compiler Output Window menu command indicates the Compiler Output window is displayed.

Note

- The Compiler Output window can be resized by clicking on a border and dragging.

{button ,AL(`ht_statusbar;ht_watch;ht_compiler;Corel_SCRIPT_Editor_windows;;',0,"Defaultoverview",)} Related Topics

To change display colors in script windows

1. Click Tools, Options.
2. Click Colors tab.
3. Click the text item whose color you want to change in the Set Colors For box.
4. Click the text color you want in the Foreground list box.
5. Click the background color you want in the Background list box.

Tip

- You can reset a text item or all text items to their original colors by clicking the Reset or Reset All buttons.

Note

- The Display Code Colors checkbox must be enabled to use display colors.

{button ,AL(`cse_settings;;;;;',0,"Defaultoverview",)} Related Topics

To change display and print fonts in script windows

1. Click Tools, Options.
2. Click Font tab.
3. Click Change button.
4. Click the font you want in the Font list box. Only monospaced fonts can be used in the Editor.
5. Click the style you want in the Font style list box.
6. Click the size you want in the Size list box.

Tip

- You can reset the font to the Corel SCRIPT Editor's original settings by clicking the Reset button instead of the Change button.

{button ,AL(`cse_settings;;;;;',0,"Defaultoverview",)} Related Topics

To change tab width in script windows

1. Click Tools, Options.
2. Click Editor tab.
3. Enter a number in the Tab Width number box. Tab width is expressed in characters.

Tip

- When the Auto Indent check box is in an enabled state, tabs are automatically inserted at the beginning a new line in a script. The tabs are inserted after you press Enter, and the number of tabs in the new line corresponds to the number of tabs in the line that precedes it.

{button ,AL(`cse_settings;;;;;',0,"Defaultoverview",)} Related Topics

To automatically save scripts before executing them

1. Click Tools, Options.
2. Click Editor tab.
3. Click the Save Scripts Before Running checkbox. A check mark in the checkbox indicates this option is enabled.
   Click checkbox again to disable this option.

Note

- If you're running a new script that has yet been saved, you will be prompted to save before execution begins.

{button ,AL(`cse_settings;;;;;',0,"Defaultoverview",)} Related Topics

To set the folders search path for INCLUDE files

To add a folder to the search path:

1. Click Tools, Options.
2. Click Folders tab.
3. Click Add.
4. To select a folder, type a path and folder name in the Folder text box or click the Browse button to select an existing folder.

To remove a folder from the search path:

1. Click Tools, Options.
2. Click Folders tab.
3. Select a folder from the folder list box and click Remove.

Notes

- The folders are search according to their order in the folder list box.
- The folder information is stored in the SCEDIT.INI file. See the **INCLUDE** statement for more information about including files in a script.

{button ,AL(`cse_settings;;;;;',0,"Defaultoverview",)} Related Topics

To prevent scripts from using implicitly declared variables

1. Click Tools, Options.
2. Click Environment tab.
3. Enable the Disable Implicit Declarations check box.

Note

- For more information about explicit declaration, click ⇨.

{button ,AL(`cse_settings;;;;;',0,"Defaultoverview",)} **Related Topics**

To execute a script from the Corel SCRIPT Editor

- Click Debug, Execute. The Execute command ignores all debugging information including script breakpoints during script execution.

Notes

- To debug a script, click Debug, Run. Running a script in debug mode in noticeably slower than running a script with the Execute command.
- You can only issue this command if a script window is active.
- If the script you're running contains Corel application commands, the script must use the **WITHOBJECT...END WITHOBJECT** statements.
- You can terminate a script's execution by pressing CTRL+BREAK.
- If you're running a new script that has yet been saved, you may be prompted to save before execution begins depending on your Corel SCRIPT Editor settings. See How to automatically save scripts before executing them for more information.

{button ,AL(`ht_open_file_cse;ht_close_file_cse;ht_debug;;;',0,"Defaultoverview",)} Related Topics

To run a script from the Corel SCRIPT Editor in debug mode

- Click Debug, Run. In debug mode, script execution stops a breakpoints or when the end of the script is reached.

Notes

- Click Debug, Execute to ignore all debugging information during a script run. Running a script in debug mode in noticeably slower than running a script with the Execute command.
- You can only issue this command if a script window is active.
- If the script you're running contains Corel application commands, the script must use the **WITHOBJECT...END WITHOBJECT** statements.
- You can also use the procedure above to restart a paused script at the current line.
- You can terminate a script's execution by pressing CTRL+BREAK.
- If you're running a new script that has yet been saved, you may be prompted to save before execution begins depending on your Corel SCRIPT Editor settings. See How to automatically save scripts before executing them for more information.

{button ,AL(`ht_open_file_cse;ht_close_file_cse;ht_debug;;;',0,"Defaultoverview",)} Related Topics

To stop the execution of a script at a breakpoint

- Click Debug, Reset.

Notes

- You can only issue this command if a script window is active.
- You can only use this command when you've paused script execution by stepping, using breakpoints, or when script execution has finished.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} Related Topics

To interrupt an executing script

- Click CTRL+BREAK.

Note

- After interrupting a script, you can step through it.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} Related Topics

To restart a script while debugging

- Click Debug, Restart.

Notes

- You can only issue this command if a script window is active.
- You can only use this command when you've paused script execution by stepping, using breakpoints, pressing ESC, or when script execution has finished.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} Related Topics

To execute a script one line at a time (Step Into)

To start stepping from the beginning of a script:

- Click Debug, Step Into. Repeat the action for each line you want to execute.

To start stepping from an intermediary position in the script.

1. Click Debug, Toggle Breakpoint where you want to begin stepping.
2. Click Run.
   Script execution is paused at the breakpoint.
3. Click Debug, Step Into. Repeat the action for each line you want to execute.

Note

- The line with the ⇨ symbol in its left margin is the next line to execute.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} **Related Topics**

To execute a script one line at a time stepping over procedures (Step Over)

To start stepping from the beginning of a script:

- Click Debug, Step Over. Repeat the action for each line you want to execute.
  When a procedure is encountered, it is executed in its entirety. Execution is then paused after the procedure call.

To start stepping from an intermediary position in the script.

1. Click Debug, Toggle Breakpoint where you want to begin stepping.
2. Click Run.
   Script execution is paused at the breakpoint.
3. Click Debug, Step Over. Repeat the action for each line you want to execute.
   When a procedure is encountered, it is executed. Execution is then paused after the procedure call.

Note

- The line with the ⇨ symbol in its left margin is the next line to execute.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} Related Topics

To execute a script to the first line after the current procedure call (Step Out)

To start stepping from the beginning of a script:

- Click Debug, Step Out.
  The debugger runs to the first line following the current procedure call and pauses.

To start stepping from an intermediary position in the script.

1. Click Debug, Toggle Breakpoint where you want to begin stepping.
2. Click Run.
   Script execution is paused at the breakpoint.
3. Click Debug, Step Out.
   The debugger runs to the first line following a procedure call and pauses.

Note

- The line with the ⇨ symbol in its left margin is the next line to execute.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} **Related Topics**

To add a variable to the Watch window

1. Place the insertion point on a variable in your script.
2. Click Add Watch

Notes

- You can also add a variable to the Watch window by entering it in the Watch window text box.
- Watches are not part of a script. Additionally, they cannot be saved and are lost when you close the script window.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} **Related Topics**

To delete a variable from the Watch window

1. Select a watch in the watch window.
2. Click  in the Watch window.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} Related Topics

To display a variable's value using the QuickWatch

1. Place the insertion point on a variable in your script.
2. Click Debug, QuickWatch.

Notes

- You can use this command only when you've paused the script's execution by stepping or using breakpoints.
- You can type in any variable in the QuickWatch window to return its current value.
- You can type a value in the Value box to change the value of the variable or expression being watched.
- Use the Convert button to change the data sub-type of a variant being watched. Before clicking, select a sub-type in the drop-down list box. Click ⇨ for more information about variants.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} **Related Topics**

To add or remove a breakpoint

1. Place the insertion point on a line to which you want to add or remove a breakpoint.
2. Click Debug, Toggle Breakpoint.

Notes

- Breakpoints cannot be saved and are lost when you close the script window.
- A line with a breakpoint has the  symbol in its left margin.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} **Related Topics**

To run a script to the cursor

1. Place the insertion point on the line where you want the script execution to stop.
2. Click Debug, Run To Cursor.

Note

- Since the insertion point acts as a breakpoint, using the Run to Cursor command is similar to using a breakpoint.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} Related Topics

To clear all breakpoints

- Click Debug, Clear All Breakpoints.

Notes

- Breakpoints cannot be saved and are lost when you close the script window.
- A line with a breakpoint has the ⇨ symbol in its left margin.
- You can only issue this command if a script window is active.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} **Related Topics**

To check a Corel SCRIPT script for syntax errors

- Click Debug, Check Syntax.

Notes

- If errors are found, error messages appear in the Compiler Output window. Double-click an error message's line number in the Compiler Output window to send the insertion point to the line containing the error. The line with the error has the ➪ symbol in its left margin after double-clicking.
- You can only issue this command if a script window is active.

{button ,AL(`ht_debug;;;;;',0,"Defaultoverview",)} **Related Topics**

To use the Call Stack window

- While stepping through a script click Tools, Call Stack.

Notes

- By clicking the Show button in the Call Stack window you can send the cursor to the position in the script where the active function or subroutine was called.
- You can only issue this command if a script window is active.

{button ,AL(`call_stack;ht_debug;;;;;',0,"Defaultoverview",)} Related Topics

To view all windows

- Click Window, Tile Horizontally or click Window, Tile Vertically.

Note

- Minimized windows are arranged at the bottom of the Corel SCRIPT Editor window.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To cascade windows in the Corel SCRIPT Editor

- Click Window, Cascade.

Note

- Minimized windows are arranged at the bottom of the Corel SCRIPT Editor window.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To close all windows in the Corel SCRIPT Editor

- Click Window, Close All.

Note

- You are prompted to save any unsaved changes in any open documents.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To arrange minimized windows

Click Window, Arrange Icons

Note

- Minimized windows are arranged at the bottom of the Corel SCRIPT Editor window.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To keep the Corel SCRIPT Editor visible in Windows

- Click Window, Always on Top. Choose the command again to turn off the setting.

Tip

- Keeping the Corel SCRIPT Editor visible, even when another application is active, is helpful when you're debugging a script.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;debugging_scripts;;',0,"Defaultoverview",)} Related Topics

To view an open window in the Corel SCRIPT Editor

- Click Window and click the window you want to view. The open windows are noted at the bottom of the Window menu.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To start online Help from Corel SCRIPT Editor

- Click Help, Help Topics.

Note

- Click the  (What's This) button and then click on an available menu command or other item on the Corel SCRIPT Editor to find out what it does.

{button ,AL(`cse_help;;;;;',0,"Defaultoverview",)} **Related Topics**

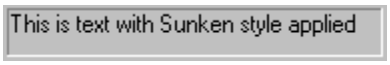
To open Corel SCRIPT online Help to a script keyword's syntax reference

1. Place the insertion point in the keyword you want help for.
2. Press F1.

Note

- If the selected keyword is not found or the insertion point is not placed in a word, Corel SCRIPT online Help displays the Help Topics dialog box.

{button ,AL(`cse_help;;;;;',0,"Defaultoverview",)} **Related Topics**


This is text with Sunken style applied

CALL statement

CALL name parameter1, parameter2, ...
name parameter1, parameter2, ...

The **CALL** statement sends script execution to a procedure. Once a called procedure has terminated execution, script execution returns to the statement that follows the statement that called the procedure. You are not required to use the **CALL** keyword when calling the procedure; however, it is good programming practice to use to **CALL** keyword to help make your scripts easier to read.

Parameter	Description
name	Name of the procedure to execute; not case-sensitive.
parameter	One or more optional arguments (variables, constants, or expressions), separated by commas, that are passed to the procedure.

Note

- In cases where the call to the procedure is placed ahead of the **SUB** or **FUNCTION** statement in a script, the procedure must first be declared with the **DECLARE** statement somewhere in the script above the call to the procedure.
- You have the option of using type-declaration characters or names in the **SUB** or **FUNCTION** statement.
- The number of parameters in a calling statement must match the number of subroutine or function parameters.
- Parenthesis are required when calling a function. Functions use the assignment operator (=). In the following example, the function **abc** assigns a value to **xyz**:

```
xyz = abc (parameter1, parameter2, ...)
```

{button ,AL(`User_defined_functions ;User_defined_subroutines;DECLARE;SUB;return;;;',0,"Defaultoverview",`main')}
Related Topics

DECLARE statement

DECLARE FUNCTION name{%|&|!|#|@|$} **(**PASS parameter{%|&|!|#|@|$}, PASS parameter AS type, ...**)** AS returnType
DECLARE SUB name (PASS parameter{%|&|!|#|@|$}, PASS parameter AS type, ...**)**

In cases where a call to a procedure (a user-defined function or subroutine) is placed ahead of the **SUB** or **FUNCTION** statement in a script, the procedure must be first made available with the **DECLARE** statement. It is programming convention to place **DECLARE** statements at the top a program or script.

Parameter	Description
FUNCTION	Used to declare a function.
SUB	Used to declare a subroutine.
name {%|&|!|#|@|$} *or* **name** ... AS returnType	Declares the name of the user-defined procedure. Optionally you can declare the data type of the value returned by the function. The data type can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option). The name and data type declaration (if exists) must correspond to the name and data type used in procedure defining statement (**FUNCTION** **SUB**).
parameter{%|&|!|#|@|$} *or* parameter AS type	Declares the procedure parameter and follows the Corel SCRIPT naming convention. Optionally you can declare the procedure parameter's data type. The data type can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option). The parameter name and data type declaration (if exists) must correspond to the parameter name and data type used in procedure defining statement (**FUNCTION** or **SUB**).
PASS	Determines how the variable it precedes in the script is passed to the procedure. PASS can be set to BYVAL or BYREF. When set to BYVAL, the value of the variable it precedes is passed by value. That is, the procedure accesses a copy of the variable and its value can't be changed by the procedure to which it was passed. When set to BYREF, the value of the variable it precedes is passed by reference and its value can be changed by the procedure. PASS is optional, and if omitted, Corel SCRIPT uses BYREF.

Note

- Parenthesis are required as shown in the syntax above.
- User-defined subroutines and functions must be declared in the main section of a script file. See Script procedures for more information.
- Use the **DECLARE...LIB** statement to call functions and subroutines in Windows dynamic link libraries (DLL files).

{button ,AL(`cs_procedures_a ;User_defined_subroutines;using_functions_subroutines;call;function_end_function;sub_end_sub;;',0,"Defaultoverview",)} Related Topics

Example for DECLARE and FUNCTION statements

```
REM main section of script file
DECLARE FUNCTION cube_function%(a%) 'function declaration
DECLARE FUNCTION rank_function(a as INTEGER, b as STRING) as STRING
'
'
MESSAGE CSTR(cube_function(3))
MESSAGE rank_function(1, "Corel Script")
'
'
REM function section of script file
FUNCTION cube_function%(a%) 'function definition
    cube_function = a^3
END FUNCTION
'
FUNCTION rank_function(a as INTEGER, b as STRING) as STRING
    rank_function = b + " is #"+ CSTR(a)
END FUNCTION
```

In the above example, two functions are declared and defined. The first function cubes an integer and the second function creates a string. The result of each function is displayed in a message dialog box. Since the calls to the functions are placed ahead of the functions in the script, the **DECLARE** statements are placed ahead of the calls.

{button ,AL(`example_vars;function_end_function;sub_end_sub;declare;;',0,"Defaultoverview",)} **Related Topics**

Example for DECLARE, SUB, and CALL statements

```
REM main section of script file
DECLARE SUB cube_sub(a%) 'sub declaration
DECLARE SUB rank_sub(a as INTEGER, b as STRING)
'
'
CALL cube_sub 3   'calling with a CALL statement
rank_sub 1, "Corel Script"   'calling without a CALL statement
'
'
REM subroutine section of script file
SUB cube_sub(a%) 'sub definition
    MESSAGE CSTR(a^3)
END SUB
'
SUB rank_sub(a as INTEGER, b as STRING)
    MESSAGE b + " is #"+ CSTR(a)
END SUB
```

In the above example, two subroutines are declared and defined. The first subroutine, **cube_sub**, displays the result of cubing the subroutine's parameter in a message dialog box. The second subroutine, **rank_sub**, displays the result of concatenating strings in a message dialog box.

The **cube_sub** is called with the **CALL** statement while **rank_sub** is not. Since the calls are placed ahead of the subroutines in the script, the **DECLARE** statements are placed ahead of the calls.

{button ,AL(`example_vars;function_end_function;sub_end_sub;declare;call;',0,"Defaultoverview",)} **Related Topics**

DECLARE...LIB statements

For declaring functions
DECLARE FUNCTION procName LIB "file" (PASS argument AS type, PASS argument AS type, ...**)** AS returnType ALIAS "aliasName"

Syntax for declaring subroutines
DECLARE SUB procName LIB "file" (PASS argument AS type, PASS argument AS type, ...**)** ALIAS "aliasName"

Corel SCRIPT scripts can be used to call procedures in Windows dynamic link libraries (DLL files).

Before you can call an external procedure, it must be declared with the **DECLARE...LIB** statement. The statement specifies a DLL, a procedure within the DLL, and the number and type of arguments passed to the procedure. Consult the DLL's technical reference before you call any of its functions. For example, to use the Windows DLLs, you need the *Windows Software Development Kit*.

Warning

- You should save or back up essential files and programs before using functions and subroutines in DLL files. Passing invalid arguments to a function can result in a Windows General Protection Fault (GPF) or unstable system behavior.

Parameter	Description
FUNCTION	Used to declare a function in a specified DLL.
SUB	Used to declare a subroutine in a specified DLL.
procName	String specifying the name of the procedure to call; must correspond to the procedure name in the DLL module where it resides; name matching is case-sensitive. You have the option of using another name for the procedure if you specify the procedure's name using the ALIAS syntax part of the statement.
"file"	String expression specifying the DDL to access. The DLL's extension and path should also be specified. If the DLL's path is not specified, Corel SCRIPT searches the system for it. Click for the Corel SCRIPT location search order.
argument	Variables that pass values to the procedure when it is called. You can use more than one argument but each argument's data type must be declared. Some procedures don't pass arguments but the brackets ( ) must still be used.
AS type	Declares an argument's data type with a type declaration name. If the procedure uses an argument, a type declaration name must be used.
ALIAS "aliasName"	String specifying the name of the called procedure; must correspond to the procedure name in the DLL module where it resides; name matching is case-sensitive. This option must be used if the DLL **procName** is not in uppercase characters. You should also use an alias if the actual name of the procedure is a reserved Corel SCRIPT keyword. Quotation marks must be used.
PASS	Determines how the variable it precedes in the script is passed to the procedure. PASS can be set to BYVAL or BYREF. When set to BYVAL, the value of the variable it precedes is passed by value. That is, the procedure accesses a copy of the variable and its value can't be changed by the procedure to which it was passed. When set to BYREF, the value of the variable it precedes is passed by reference and its value can be changed by the procedure. PASS is optional, and if omitted, Corel SCRIPT uses BYREF. If a procedure expects a BYREF parameter, it cannot be passed a non-variable (for example, a constant) since BYREF indicates that the called procedure can modify the contents of what you pass. Use BYVAL in all parameters that you do not expect the called function to modify.
AS returnType	Declares the data type of the value returned by the function with a type declaration name.

Note

- You can declare an argument's data type using a Corel SCRIPT type declaration character, for example, X$ instead of X AS STRING
- The **DECLARE...LIB** statement can be placed anywhere in a script before the first call to the declared procedure.
- You might not be able to use a DLL procedure that uses a variable type that is not supported by Corel SCRIPT. See Corel SCRIPT data type summary for a list of supported data types. Declare unsupported data types as a Corel SCRIPT data type if the unsupported data type uses the same number of bytes as a Corel SCRIPT data type. For example, most Win32 procedures expect LONG (4 byte) integers, therefore the Core SCRIPT data type INTEGER (2 byte) will not work for these procedures. In these cases, change **Parameter as INTEGER** to **Parameter as LONG**.
- The "A" character must be added to the end of alias with almost all Windows procedures using string parameters. The common name of a procedure doesn't really exist in the Windows DLL. Instead, there are in two versions of these Windows procedures. For example the **FindWindow** procedure has two versions one for ANSI strings (**FindWindowA**) and one for

UNICODE strings (**FindWindowW**). Since Corel SCRIPT uses ANSI strings, you must use the ANSI version of the procedure.

- If you declare a procedure wrongly, for instance, use INTEGER in place of LONG, you may get unexpected results (even General Protection Faults) while running the script. This is because Corel SCRIPT expects the declaration to be correct; there is no way of verifying the types of parameters that a DLL procedure expects with what is declared.

{button ,AL(`Using_Dynamic_Link_Libraries;getapphandle;getwinhandle;declare;function_end_function;sub_end_sub;call;Corel_SCRIPT_advanced_programming_features;',0,"Defaultoverview",)} **Related Topics**

Corel SCRIPT location search order

Corel SCRIPT searches for DLLs specified with the **DECLARE...LIB** statement and the SCPINT**xx**.DLL (run-time interpreter file where **xx** indicates the major version number) used with Corel SCRIPT Executables in the following locations and order:

1. The folder from which the script or executable was loaded.
2. The active folder.
3. The system subfolder in the Windows folder for Windows 95 and Windows NT.
4. The Windows folder.
5. The folders listed in the system's PATH environment variable. Type PATH in an MS-DOS window for a list.

For more information about version numbers see the **GETVERSION** function.

Examples of DECLARE...LIB statement

```
DECLARE FUNCTION GetActiveWindow LIB "C:\WINDOWS\user.exe" ( ) AS INTEGER
DECLARE SUB CloseWindow LIB "C:\WINDOWS\user.exe" (win AS INTEGER)
```

The first statement defines a procedure named **GetActiveWindow** (a Windows function that does not require a type-declaration character). The executable code for this procedure is stored in "user.exe" which is actually a dynamic link library without a DLL extension. The brackets ( ) indicate an empty parameter list, and the keywords **AS INTEGER** describes the type of value the function returns.

The second statement is similar, except that a **SUB** procedure does not return a value. The parameter's data type is declared with an **AS** keyword, as shown by **(win AS INTEGER)**.

The following are some examples of the **DECLARE...LIB** statement using Windows DLL files:

```
DECLARE FUNCTION FindWindow LIB "user32" (BYVAL classname AS LONG,BYVAL title AS STRING) AS
INTEGER ALIAS "FindWindowA"
```
Looks for a given window title and class. Returns the handle of that window.

```
DECLARE FUNCTION SetFocus LIB "user32" (BYVAL hwnd AS INTEGER) AS INTEGER ALIAS "SetFocus"
```
Gives focus to a specified window.

```
DECLARE FUNCTION WinExec LIB "kernel32" (BYVAL path AS STRING,BYVAL shw AS INTEGER) AS INTEGER
ALIAS "WinExec"
```
Executes the specified executable (.exe).

```
DECLARE FUNCTION GetActiveWindow LIB "user32" () AS INTEGER ALIAS "GetActiveWindow"
```
Returns the handle of the active window.

```
DECLARE SUB CloseWindow LIB "user32" (BYVAL win AS INTEGER) ALIAS "CloseWindow"
```
Closes the specified window.

{button ,AL(`declare;function_end_function;sub_end_sub;call;Corel_SCRIPT_advanced_programming_features;',0,"Defaultoverview",)} **Related Topics**

FUNCTION and END FUNCTION statements

FUNCTION name{%|&|!|#|@|$} **(**PASS parameter{%|&|!|#|@|$}, PASS parameter AS type, ...**)** AS type
    [statements]
END FUNCTION

The **FUNCTION** statement is the first line of a user-defined function and **END FUNCTION** is the last line. A function is a series of Corel SCRIPT statements that are executed to return a single value. You should use subroutines and functions when you want to repeat a series of Corel SCRIPT statements in a script. Using these user-defined procedures can make your scripts easier to change and debug.

You use, or call, a user-defined function the same way that you call any other Corel SCRIPT programming function. Calling statements consist of a function name, which can be followed by one or more parameter values that are passed to the function. If there are no parameters, empty parentheses must follow the function name. The number of parameters in a calling statement must match the number of function parameters. In cases where the call to function is placed ahead of the **FUNCTION** statement in a script, the function must be first declared with the **DECLARE** statement. Functions do not execute unless they are called.

For more information about functions, see Script Procedures.

Parameter	Description
name{%|&|!|#|@|$} or **name**...AS TYPE	Specifies the name assigned to the function. It follows the Corel SCRIPT naming convention. Optionally, you can declare the data type returned by the function. The data type can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option). If the type is not specified, the function returns a variant.
parameter{%|&|!|#|@|$} *or* parameter AS type	Specifies the variable(s) that store the value(s) that is passed to the function. The variables follow the Corel SCRIPT naming convention. Optionally, you can declare the data type of the variable. The data type can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option). If the type is not specified, the parameter is a variant.
[statements]	Script instructions that are executed when the function is called. The instructions may return a value, but it is not necessary.
PASS	Determines how the variable it precedes in the script is passed to the procedure. PASS can be set to BYVAL or BYREF. When set to BYVAL, the value of the variable it precedes is passed by value. That is, the procedure accesses a copy of the variable and its value can't be changed by the procedure to which it was passed. When set to BYREF, the value of the variable it precedes is passed by reference and its value can be changed by the procedure. PASS is optional, and if omitted, Corel SCRIPT uses BYREF.

Note

- Parameters are optional in the **FUNCTION** statement but the parentheses are required.
- It's a generally accepted programming convention to indent function statements.
- Use the **DECLARE...LIB** statement to declare functions and subroutines in Windows dynamic link libraries (DLL files).
- Your function and subroutines procedures should be self-contained; that is, a variable only required within a procedure should be a local or static variable. Following this advice can make your procedures more modular, enabling you to copy them to other scripts with limited customization.
- A function cannot be defined inside another function or subroutine.
- Functions can be called from a function, or functions can be called recursively. For an example of a recursive function click .
- You can immediately exit a function with the **EXIT** statement. Script execution is sent to the script instruction which follows the call to the function.

{button ,AL(`cs_procedures_a ;Script_procedures ;using_functions_subroutines;declare;static;call;sub_end_sub;;;',0,"Defaultoverview",)} Related Topics

Example of recursive function

```
REM Example of factorial function

FUNCTION facto(BYVAL a%) AS INTEGER
    IF a%>1 THEN
        facto = facto(a-1)*a
    ELSE
        facto = 1
END FUNCTION


x% = facto(4)
message x
```

In the above example, the **facto** function is used to calculate a factorial. For example, the factorial of 4 equals 4*3*2*1 or 24. The above function multiplies the parameter by (parameter less 1) until the (parameter less 1) equals 1. The result, 24, is displayed in a message box.

The **facto** parameter is passed by value (BYVAL).

{button ,AL(`example_vars;function_end_function;sub_end_sub;declare;;',0,"Defaultoverview",)} Related Topics

SUB...END SUB statements

SUB name (parameter{%|&|!|#|@|$}, parameter AS type, ...)
    [statements]
END SUB

The **SUB** statement is the first line of a user-defined subroutine and **END SUB** is the last line. A subroutine is a series of Corel SCRIPT statements that are executed when called. Unlike functions, subroutines don't return a value. You should use subroutines and functions when you want to repeat a series of Corel SCRIPT statements in a script. Using these user-defined procedures can make your scripts easier to change and debug. Subroutines cannot execute unless called.

To use or call a subroutine, see the **CALL** statement. In cases where the call to subroutine is placed ahead of the **SUB** statement in a script, the subroutine must be first declared with the **DECLARE** statement. For more information about subroutines, see Script Procedures.

Parameter	Description
name	Specifies the name assigned to the subroutine. It follows the Corel SCRIPT naming convention.
parameter{%|&|!|#|@|$} *or* parameter AS type	Specifies the variable(s) that store the value(s) that is passed to the subroutine. The variables follow the Corel SCRIPT naming convention. Optionally, you can declare the data type of the variable. The data type can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option).
[statements]	Script instructions that are executed when the subroutine is called.
PASS	Determines how the variable it precedes in the script is passed to the procedure. PASS can be set to BYVAL or BYREF. When set to BYVAL, the value of the variable it precedes is passed by value. That is, the procedure accesses a copy of the variable and its value can't be changed by the procedure to which it was passed. When set to BYREF, the value of the variable it precedes is passed by reference and its value can be changed by the procedure. PASS is optional, and if omitted, Corel SCRIPT uses BYREF.

Note

- Subroutines and functions are both Corel SCRIPT procedures that execute instructions but functions can also return values.
- Use the **DECLARE...LIB** statement to declare functions and subroutines in Windows dynamic link libraries (DLL files).
- Your function and subroutines procedures should be self-contained; that is, a variable only required within a procedure should be a local or static variable. Following this advice can make your procedures more modular, enabling you to copy them to other scripts with limited customization. Subroutine variables are local to the subroutine. A variable with the same name as a subroutine variable can be used elsewhere in the script without conflict.
- The number of parameters in a subroutine calling statement must match the number of subroutine parameters.
- A subroutine cannot be defined inside another subroutine.
- Subroutines can be called from a subroutine, or a subroutine can be called recursively.
- Subroutines can include line number and line labels which are not visible outside the subroutine. A line label inside a subroutine should not have the same name as a subroutine or function. For more information about line numbers and labels see the **GOTO** statement.
- It's a generally accepted programming convention to indent subroutine statements.
- You can immediately exit a subroutine with the **EXIT** statement. Script execution is sent to the script instruction which follows the call to the subroutine.

{button ,AL(`cs_procedures_a ;Script_procedures ;function_end_function;using_functions_subroutines;declare;call;;;', 0,"Defaultoverview",)} Related Topics

Corel SCRIPT character map

Corel SCRIPT uses the American National Standards Institute (ANSI) character mapping. At 256 characters, the ANSI character set offers twice as many characters as ASCII text, including special characters such as the copyright symbol, accents, and mathematical and currency symbols.

The first 128 characters (numbers 0-127) in the ANSI set correspond to characters on a standard U.S. keyboard, and include many non-printing characters. The second 128 characters (numbers 128-255) return different characters, depending on the font used.

Because some character symbols in Corel SCRIPT are reserved for use in syntax, such as quotation marks for defining strings, the CHR function and ANSI character codes should be used to include special symbols in a script. For example, to add double quotation marks to a string variable, you use ANSI character 34:

```
s$ = CHR(34) + "This will be in double quotes." + CHR(34)
MESSAGE s$
```

You can also use the function to add a return and a line feed within a string, with character 13 and 10, respectively:

```
s$ = "String 1" + CHR(13) + CHR(10) + "String 2"
MESSAGE s$
```

The result will be the two strings on separate lines, as displayed in the message box.

Character Set (ANSI 0 - 127)

Character Set (ANSI 128 - 255)

{button ,AL(`using_strings;chr;cs_character_map;;;',0,"Defaultoverview",)} Related Topics

Character Set (ANSI 0 - 127)

0	☒	32	[SPACE]	64	@	96	`
1	➪	33	!	65	A	97	a
2	➪	34	"	66	B	98	b
3	➪	35	#	67	C	99	c
4	➪	36	$	68	D	100	d
5	➪	37	%	69	E	101	e
6	➪	38	&	70	F	102	f
7	➪	39	'	71	G	103	g
8	[BACKSPACE]	40	(	72	H	104	h
9	[TAB]	41	)	73	I	105	i
10	[LINEFEED]	42	*	74	J	106	j
11	➪	43	+	75	K	107	k
12	➪	44	,	76	L	108	l
13	[RETURN]	45	-	77	M	109	m
14	➪	46	.	78	N	110	n
15	➪	47	/	79	O	111	o
16	➪	48	0	80	P	112	p
17	➪	49	1	81	Q	113	q
18	➪	50	2	82	R	114	r
19	➪	51	3	83	S	115	s
20	➪	52	4	84	T	116	t
21	➪	53	5	85	U	117	u
22	➪	54	6	86	V	118	v
23	➪	55	7	87	W	119	w
24	➪	56	8	88	X	120	x
25	➪	57	9	89	Y	121	y
26	➪	58	:	90	Z	122	z
27	➪	59	;	91	[	123	{
28	➪	60	<	92	\	124	|
29	➪	61	=	93	]	125	}
30	➪	62	>	94	^	126	~
31	➪	63	?	95	_	127	➪

Note

- ➪ Characters displayed with a ➪ do not have a graphical representation in ANSI.

Character Set (ANSI 128 - 255)

{button ,AL(`using_strings;chr;cs_character_map;;;',0,"Defaultoverview",)} Related Topics

Character Set (ANSI 128 - 255)

128	➪	160	[SPACE]	192	À	224	à
129	➪	161	¡	193	Á	225	á
130	➪	162	¢	194	Â	226	â
131	➪	163	£	195	Ã	227	ã
132	➪	164	¤	196	Ä	228	ä
133	➪	165	¥	197	Å	229	å
134	➪	166	¦	198	Æ	230	æ
135	➪	167	§	199	Ç	231	ç
136	➪	168	¨	200	È	232	è
137	➪	169	©	201	É	233	é
138	➪	170	ª	202	Ê	234	ê
139	➪	171	«	203	Ë	235	ë
140	➪	172	¬	204	Ì	236	ì
141	➪	173		205	Í	237	í
142	➪	174	®	206	Î	238	î
143	➪	175	¯	207	Ï	239	ï
144	➪	176	°	208	Ð	240	ð
145	‘	177	±	209	Ñ	241	ñ
146	’	178	²	210	Ò	242	ò
147	➪	179	³	211	Ó	243	ó
148	➪	180	´	212	Ô	244	ô
149	➪	181	µ	213	Õ	245	õ
150	➪	182	¶	214	Ö	246	ö
151	➪	183	·	215	×	247	÷
152	➪	184	¸	216	Ø	248	ø
153	➪	185	¹	217	Ù	249	ù
154	➪	186	º	218	Ú	250	ú
155	➪	187	»	219	Û	251	û
156	➪	188	¼	220	Ü	252	ü
157	➪	189	½	221	Ý	253	ý
158	➪	190	¾	222	Þ	254	þ
159	➪	191	¿	223	ß	255	ÿ

Note

- ➪ Characters displayed with a ➪ do not have a graphical representation in ANSI.

Character Set (ANSI 0 - 127)

{button ,AL(`using_strings;chr;cs_character_map;;;',0,"Defaultoverview",)} Related Topics

Reserved keywords

The following keywords are reserved by Corel SCRIPT and cannot be used as variable, constant, procedure, function, subroutine, parameter, or array names:

ABS
ACOS
ADDFOL
ADDITEM
ADDRESBIN
ADDRESBMP
ADDRESICO
ALIAS
AND
ANGLECONVERT
APPEND
AS
ASC
ASIN
AT
ATAN

BEEP
BEGIN
BEGINWAITCURSOR
BITMAPBUTTON
BOOLEAN
BUILDDATE
BUILDTIME
BYREF
BYVAL

CALL
CANCELBUTTON
CASE
CBOL
CCUR
CDAT
CDBL
CHECKBOX
CHR
CINT
CLNG
CLOSE
CLOSEDIALOG function
COMBOBOX
CONST
COPY
COS
CREATEOBJECT
CSNG
CSTR
CURRENCY

DATA
DATE
DDCOMBOBOX
DDLISTBOX
DEC
DECLARE
DEFINE
DIALOG
DIM
DO
DOUBLE

ELSE
ELSEIF
ENABLE
END
ENDIF
ENDWAITCURSOR
EOF
EQV
ERROR
EXIT
EXP

FAIL
FALSE
FIELD
FILEATTR

FILEDATE
FILEMODE
FILEPOS
FILESIZE
FINDFIRSTFOLDER
FINDNEXTFOLDER
FIX
FOR
FORMATDATE
FORMATTIME
FREEFILE
FROMCENTIMETERS
FROMCICEROS
FROMDIDOTS
FROMINCHES
FROMPICAS
FROMPOINTS
FUNCTION

GET
GETAPPHANDLE
GETBITMAPHEIGHT
GETBITMAPWIDTH
GETCOLOR
GETCOMMANDLINE
GETCURRDATE
GETCURRFOLDER
GETDATEINFO
GETFILEBOX
GETFOLDER
GETFONT
GETHEIGHT
GETHELPINDEX
GETHELPPATH
GETIMAGE
GETID
GETINCREMENT
GETITEM
GETITEMCOUNT
GETLEFTPOSITION
GETMAXRANGE
GETMINRANGE
GETOBJECT
GETPRECISION
GETPROCESSINFO
GETSCRIPTFOLDER
GETSELECT
GETSTYLE
GETTEMPFOLDER
GETTEXT

GETTICK
GETTIMEINFO
GETTIMER
GETTOPPOSITION
GETTYPE
GETVALUE
GETVERSION
GETWIDTH
GETWINHANDLE
GLOBAL
GOSUB
GOTO
GROUPBOX

HELPBUTTON
HEX
HSLIDER

IF
IMAGE
IMAGELISTBOX
IMP
INCLUDE
INPUT
INSTR
INT
INTEGER
IS

KILL

LBOUND
LCASE
LEFT
LEN
LENGTHCONVERT
LET
LIB
LINE
LISTBOX
LN
LOCK
LOF
LOG
LONG
LOOP
LSET
LTRIM

MESSAGE

MESSAGEBOX
MID
MKFOLDER
MOD
MOVE

NEXT
NOT
NOTHING

OBJECT
OKBUTTON
ON
OPEN
OPTIONBUTTON
OPTIONGROUP
OR
OUTPUT

PRESERVE
PRINT
PROGRESS
PUSHBUTTON
PUT

RANDOMIZE
READ
REDIM
REGISTRYQUERY
REM
REMOVEITEM
RENAME
RESET
RESTORE
RESUME
RETURN
RIGHT
RMFOLDER
RND
RTRIM

SEEK
SELECT
SET
SETBITMAPOFFSET
SETCURRDATE
SETDOUBLEMODE
SETEMPTY
SETHELPINDEX
SETHELPPATH

SETIMAGE
SETINCREMENT
SETMAXRANGE
SETMINRANGE
SETPRECISION
SETSELECT
SETSTYLE
SETTEXT
SETTHREESTATE
SETTICK
SETTIMER
SETVALUE
SETVISIBLE
SGN
SIN
SINGLE
SPACE
SPINCONTROL
SQR
STARTPROCESS
STATIC
STATUS
STEP
STOP
STRING
STR
SUB
SWAP

TAN
TEXT
TEXTBOX
THEN
TIMER
TO
TOCENTIMETERS
TOCICEROS
TODIDOTS
TOINCHES
TOPICAS
TOPOINTS
TRUE
TYPE

UBOUND
UCASE
UNDEF
UNLOCK
UNTIL

VAL
VARIANT
VSLIDER

WAIT
WEND
WHILE
WITH
WITHOBJECT
WRITE

XOR

This word is reserved for future use.

This word used in Corel SCRIPT version 6.0.

Corel SCRIPT glossary

Add-ons
Alignment
Align
ANSI
Application command
Application function
Array
Assignment Statements
ASCII

Binary
Bitmap
BMP
Boolean variable
Bounding box
Breakpoint
By reference (BYREF)
By value (BYVAL)

Character code
Check box
Clipboard
Comment statements
Compile
Conditional statements
constant
Corel SCRIPT
Corel SCRIPT Binary
CSB

Current folder
Cursor
Custom dialog box

Data type
Debug
Declare
Dialog box
Dialog unit
Didot
DLL (dynamic link library)
Dynamic dialog boxes

Executable
Expression
External scripts

Folder
Function

Global
Group

Handles

Internal scripts

List boxes
Local
Loop

Marquee
Maximize
Micron
Minimize

Numeric expression

OLE Automation
Operators
Option buttons

Parameters
Pica
Point
Procedure
Programming statements
Prompt

Recording

String
String expression
Subroutine
Syntax

Trace
Type-declaration character

Variable

Wildcard
Window
Window handle

Add-ons

Corel Add-ons are programs that add custom features to Corel applications. Third-party software developers are making Add-ons commercially available to meet the specialized needs of Corel users. Currently, Corel VENTURA 7 is the only Corel application that supports Corel Add-ons.

Alignment, relative

Setting two or more objects to some meaningful spatial relationship, such as centering them or distributing them evenly along a line.

Alignment, text

Text alignment affects text objects of more than one line. The lines of text may be aligned to the left edge, center, or right edge of the text frame.

ANSI

The American National Standards Institute character set. It consists of 256 characters; the first 128 are the same as the ASCII character set.

Application command

Application commands perform tasks in a specific Corel application. For example, the **FileNew** command creates a new document. Each Corel application that supports Corel SCRIPT has a distinct set of commands. Any script you create by saving a recording of your actions is made of Corel SCRIPT application commands.

Application command names often describe the action they perform since most are one-word equivalents of the corresponding Corel application user-interface. For example, the **EditCut** command is the complement of the Cut command found in the Edit menu.

Application function

Application functions ask questions about the status of a Corel application, selected items in Corel applications, or document properties. For example, a function may ask a Corel application about a document's page size. Functions cannot be recorded.

Array

An ordered set of items of the same data type. The items can be referred to both as a unit and individually. Click  for more information.

Assignment statements

Assign the value of an expression to a variable.

```
x = "John Doe"

REM x equals John Doe

y = 5

REM y equals 5

z = 3 + 4

REM z equals the result of 3 + 4
```

The assignment operator (=) assigns the value of a right operand expression to a left operand variable. Click  for more information.

ASCII

The American Standard Code for Information Interchange character set which consists of the characters available on a standard 128 character keyboard. ASCII also includes non-printable control codes such as carriage returns and page breaks.

Corel SCRIPT Binary (.CSB)

A Corel SCRIPT script compiled into computer binary format. A script compiled into binary format does not have to be re-compiled each time it is run. Click ⇨ for more information.

Binary also means a system of counting based on a unit of two.

Bitmap

An image composed of a series of pixels or dots. Scanners and paint programs, such as Corel PHOTO-PAINT and Microsoft Paintbrush, generate this type of image. By contrast, CorelDRAW creates images using vector objects⇨shapes stored internally as mathematical equations.

BMP

The filename extension for Windows bitmap files.

Boolean

A Corel SCRIPT data type which can only equal TRUE (-1) or FALSE (0). Click  for more information.

Bounding box

A box drawn around an object or group. A bounding box is the smallest rectangular box in which the object (or group) will fit completely. The bounding box is parallel to the axes of the object.

Breakpoint

A line in a script at which execution pauses, usually to aid in debugging. Click  for more information.

By reference (BYREF)

A method of passing an argument to a procedure that enables the procedure to change the argument's value.

By value (BYVAL)

A method of passing an argument to a procedure that enables the procedure to alter a copy of the argument but not change the value of the original argument.

Character code

The number corresponding to a character in a character set, such as the ASCII or ANSI character sets. Click ⇨ for more information.

Check box

A square box in a dialog box used to enable and disable options. An option is enabled when an X or a check mark appears in the check box, and is disabled when the check box is empty.

Clipboard

A temporary storage area used to hold cut or copied information. You can paste the contents of the Clipboard into other programs that support the type of information that has been cut or copied. Information remains on the Clipboard until another cut or copy command is executed.

Comment statements

Contain notes and other information that do not affect script execution. Use comment statements to explain the purpose of your script, to describe its components, or to prevent a statement from playing. Comment statements are helpful when you have to modify a script months after it is written, or when someone else is attempting to understand your script. A comment begins an apostrophe or a **REM**.

Compile

The period of time during which a script is translated into executable code.

Conditional statements

A statement or group of statements (statement block) that execute when a specified condition is met. Conditional statements are useful for displaying a list of options. Conditional statements include CASE, IF, and SWITCH. Click  for more information.

Constant

A name given to a particular value, such as pi (3.14152), or a name, such as MyCompany. A constant's value cannot be altered during script execution.

Corel SCRIPT

A programming language used to create programs to use with Corel applications such as CorelDRAW or Corel VENTURA.

Current Folder

The folder (or directory) where a file will be saved if no other directory is specified.

Cursor

Also called the mouse pointer. It indicates the object, command, tool, or other screen item you want to select. The shape of the cursor changes depending on the tool or command you have selected.

Custom dialog box

A dialog box created with Corel SCRIPT programming statements and functions that displays options for user input. Click ➪ for more information.

Data type

Defines the set of values that a variable can store. For more information about data types, click ⇨.

Debug

To remove the syntax and logic errors that prevent the script from either being compiled or executing correctly. Click  for more information.

Declare

A statement (**DIM**) that states the name and data type of a variable to be used in a script.

Dialog box

A window that displays warnings, messages, and options for users to select. Dialog boxes have a title bar and a control menu, but not a menu bar. They can be moved to different positions on the screen.

Dialog unit

The dialog unit measurement is used in Corel SCRIPT dialog boxes. For width, a dialog unit is 1/4 the average width of the Corel system font. For height, a dialog unit is 1/8 the average height of the Corel system font. In other words, a dialog unit for width and height are practically equal because, on average, the height of the Corel system font is twice its width (1/8 x 2=1/4). Creating a dialog box that is 200 units (width) by 200 units (height) results in a dialog box that is a square or very close to a square.

Didot

A unit of measure equivalent to 1.07 times a U.S. point. Sixty seven thousand, five hundred sixty seven (67.567) didots equal one inch.

DLL (dynamic link library)

A library of functions and subroutines that can be called from a script at run time.

Dynamic dialog boxes

Unlike static dialog boxes, dynamic dialog boxes can change their display contents based on user action such as clicking a push button or selecting a list item. Most Windows applications use dynamic dialog boxes. For example, click Tools, Options in the Corel SCRIPT Editor to open the Options dialog box: clicking on a tab changes the dialog box appearance. Click ➪ for more information.

Executable

Corel SCRIPT scripts are text files and do not have a compiled executable component. Before a script is executed, it is internally compiled into a binary file, and is re-compiled each time the script is run. To save the compile time, you can compile your scripts into Corel SCRIPT Executables. Compiling your scripts into executables not only speeds up their run-time, but allows you to hide the programming code that has gone into creating the script.

Expression

Numeric expression: a combination of numbers, variables, constants, functions, and operators that return a number.

String expression: a combination of literals, string variables, string constants, and string operators that return a string.

External scripts

Corel VENTURA supports two types of scripts: external and internal. External scripts are ordinary text files that are saved to disk as a separate file. Internal scripts are embedded in a publication and can only be executed or edited if the publication has been opened.

Folder

Folders are used to store and organize your documents, programs, and other files. For example, you could create a folder called LOGOS for storing logo designs. In previous versions of Windows, folders were called directories.

Function

Functions are a group of Corel SCRIPT statements and commands that are treated as a block. Functions are accessible from other parts of a script, and can be called any number of times during execution. A function can also return a value to an executing script.

To define a functions, use the FUNCTION...END FUNCTION statements.

Global

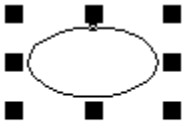
Global variables are available anywhere in a running script, but they and their values cease to exist when the script stops running. Global variables are created in the main section of a script and cannot be created within a subroutine or a function. However, they can be used in the execution of any subroutine or function. Use the GLOBAL statement to create global variables.

Group

A set of collected objects. Grouping enables a set of simple objects to behave as one object.

Handles

Small squares that appear on the corners and sides of an object's highlighting box when the object is selected. Use the square handles to resize and transform an object. Click on a selected object and the handles change to arrows which then permit you to rotate and skew the object.



Sizing handles

Internal scripts

Corel VENTURA supports two types of scripts: external and internal. External scripts are ordinary text files that are saved to disk as a separate file. Internal scripts are embedded in a publication and can only be executed or edited if the publication has been opened.

List boxes

Display a list of options. There are different styles of list boxes available in Corel SCRIPT, See **LISTBOX**.

Local

Local variables are available in the procedure in which they are declared. If declared in a subroutine or function, a local variable ceases to exist after the procedure finishes execution and is re-created the next time the subroutine or function is called.

Loop

Statement or group of statements (statement block) that executes a specified number of times until an expression is true or while an expression is true. The script then exits the loop and continues to the next statement. Loop statements include **Do...Loop**, **For...Next**, and **While...Wend**. Click  for more information.

Marquee

The rectangle that results when you drag diagonally to select a set of elements or a region on an image, the rectangle created between the start point (mouse down) and the end point (mouse up) is called a marquee.

Maximize

To enlarge an application window to full-screen size.

Minimize

To reduce an application window to an icon in the docking bar.

Micron

A unit of measure equivalent to one millionth of a meter. CorelDRAW and Corel VENTURA use tenths of a micron as standard unit of measure in scripts. See the **LENGTHCONVERT** statement to convert microns to other units of measure.

Unit of measurement	Number of tenths of a micron per unit
inch	254,000
centimeters	100,000
points	3,527.78
ciceros	45,118.7
didots	3,759.2
picas	42,333.33

Numeric expression

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

OLE Automation

OLE Automation is an integration standard that allows applications to expose their programmable objects so that other applications can control them. Exposing an object means an application makes the script or macro commands that control it available to other programming applications. The exposed commands become an extension of the controlling programming language. Click  for more information.

Operators

A symbol or word that performs a function on one or more expressions. Operators compare expressions, link words together, and perform mathematical functions. Click ⇨ for more information.

Option buttons

Display mutually exclusive options. Option buttons are also called radio buttons.

Parameters

Parameters pass information to Corel SCRIPT statements, commands, or functions so that they can perform a specified action. In this Corel VENTURA example, .**FileOpen** is the command name, and **"C:\VENTURA\MYFILES\TEST1.VP"** is a parameter which specifies a file to open.

```
.FileOpen "C:\VENTURA\MYFILES\TEST1.VP"
```

Pica

A unit of measurement used primarily in typesetting. One pica equals approximately 1/6 of an inch (exactly 12 points).

Point

A unit of measure used primarily in typesetting for designating type sizes. There are approximately 72 points(pts) to an inch and exactly 12 points to a pica.

10 pts 18 pts 36 pts

Procedure

Corel SCRIPT scripts are comprised of three types of procedures:

- main section of a script (one per script)
- user-defined functions (optional and more than one can exist)
- user-defined subroutines (optional and more than one can exist)

If you have a group of instructions that will be repeated in a script, create a user-defined procedure for those instructions. The instructions are written once in the script and can be called from different places within the script. If the instructions are changed, the changes take effect everywhere. Using these user-defined procedures can make your scripts easier to change and debug.

Programming statements

Corel SCRIPT programming statements and functions are a common set of instructions that can be used with any Corel application that supports scripting. Programming statements and functions are derived from traditional BASIC programming language dialects and perform instructions that are not part of a Corel application. In the Corel SCRIPT online Help, Corel SCRIPT programming statements and functions appear in uppercase, for example, LEFT, IF, and MESSAGEBOX. Click  for a listing of all programming statements and functions.

Prompt

A message box that displays information for the user.

Recording

A series of commands and keystrokes saved to a portion of an application's memory called a "recording." Playing a recording results in commands and keystrokes being repeated.

String

A data type that accepts characters enclosed in double quotation marks. Click ➪ for more information.

String expression

In Corel SCRIPT, a string expression is a combination of literals, string variables, string constants, and string operators that return a string. Click ⇨ for more information.

Subroutine

Subroutines are a group of Corel SCRIPT statements and commands that are treated as a block. Subroutines are accessible from other parts of a script, and can be called any number of times during execution.

To define a subroutine, use the SUB...END SUB statements.

Syntax

The set of rules that determine the order and format in which a command, statement, or function is made in order for it to be understood by the Corel SCRIPT compiler.

Refers to rules that govern the form of macro statements and expressions. For example,

MESSAGE "John Doe"

displays a message box with John Doe in it. The next example,

MESSAGE "John Doe"

lacks a closing quotation mark. The syntax is not correct and produces an error message. Click ⇨ for more information.

Trace

To follow the progression of the script, line-by-line, as it executes. Click ⇨ for more information.

Type-declaration character

The character suffixed to a variable name, indicating the data type of the variable. Click  for more information.

Variable

Represent data that can change during script execution. Data may include any type of expression, but only one expression at a time. Variable names are user defined, are not case sensitive, must begin with a letter, can include any other combination of letters or numbers. Click ⇨ for more information.

Wildcard

Characters that represent variables in a word, file, or directory search. A question mark (?) represents a single character. An asterisk (*) represents zero or more characters in succession.

Window

A rectangular area on the screen in which applications are displayed. Every application window has a title bar and menu bar along the top and one or two scroll bars along the sides or bottom.

Window Handle

A unique internal identifier for a window or control. For more information, see the **GETWINHANDLE** function.

ASC function

ASC(source)

Returns the numerical ANSI character value of the first character specified in a string. ASC is the opposite of the **CHR** function, which returns a character when the ANSI value is specified.

Parameter	Description
source	The string expression to be examined.

Note

- See the Corel SCRIPT Character Map for more ANSI details and Windows characters.

Example

```
i% = ASC("string")
```

This expression will assign the value 115, which is the ANSI value of the letter "s."

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

CHR function

CHR (value)

Returns the ANSI character that occupies the specified position in the ANSI character set. **CHR** is the opposite of **ASC**, which returns the ANSI code value when the character is entered. See the Corel SCRIPT Character Map for more ANSI details and Windows characters.

Parameter	Description
value	ANSI code value to be examined.

Example

```
s$ = CHR(65)
```

Assigns the letter "A" to the variable **s$**. (Character 65 of the ANSI character set is A.)

Special Characters

The CHR function is often used to add special characters to string variables that cannot be entered directly within double quotation marks. For example, to add double quotation marks to a string, you use character 34:

```
s$ = CHR(34) + "This will be in double quotes." + CHR(34)
MESSAGE s$
```

You can also use the function to add a return and a line feed within a string; use character 13 and 10, respectively:

```
s$ = "String 1" + CHR(13) + CHR(10) + "String 2"
MESSAGE s$
```

This will place the two strings on separate lines, as displayed in the message box.

The following table notes some of the special characters you can use with the **CHR** function.

Character Number	Special Character Returned
8	Backspace
9	Tab
10	Linefeed
13	Return
32	Space
34	Quotation mark

The following table notes some of the special characters you can use with the **CHR** function if you're using Corel VENTURA commands .**InsertSymbol** or .**TypeText**.

Character Number	Special Character Returned
10	Paragraph return
13	Forced line break
17	Em space
18	En space
19	Figure space
20	Thin space
21	Non-breaking space
22	Discretionary hyphen
34	Straight double quote
145	Typographical open single quote
146	Typographical close single quote
147	Typographical open double quote
148	Typographical close double quote
150	En dash
151	Em dash
153	Trademark
169	Copy right
174	Registered mark

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

INSTR function

INSTR (string1, string2, start**)**

Returns the starting position of the first occurrence of a string within another string. If the specified string is not found, the function returns 0.

Parameter	Description
string1	The string expression within which the search is made.
string2	The string for which you are searching.
start	Lets you specify the position where the search begins within **string1**. If unspecified, the search starts at the beginning of **string1** (same as **start**=1). Must be a non negative number and fractional numbers are rounded.

Example

```
pos = INSTR("Los Angeles", "Ang")
```

Sets **pos** to the value 5 because "Ang" occurs at the fifth character in the string "Los Angeles".

```
pos = INSTR("Los Angeles: City of Angels", "Ang", 8)
```

Sets **pos** to the value 22.

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

LCASE function

LCASE(source)

Converts a string to lowercase characters.

Parameter	Description
source	The string expression to convert.

Note

- Non letter characters do not change when this function is used.
- You can use **UCASE** to convert to uppercase characters.

Example

```
x$="HI"
firststring$ = LCASE(x)
secondstring$ = LCASE("THerE")
MESSAGE firststring + " " + secondstring
```

The above example displays the converted strings "hi there" in a message box.

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

LEFT function

LEFT (source, number)

Returns the specified number of characters from the beginning of a string.

Parameter	Description
source	The string expression from which the specified characters are returned.
number	Lets you specify the number of characters to be returned. Must be a non negative number and fractional numbers are rounded.

Note

- This function can be used to truncate user input from a dialog box.
- You can use the **LEN** function to determine the number of characters in a string.
- You can use the **RIGHT** function to return characters from the end of a string.

Example

```
abc$ = LEFT("I want to dance with you", 15)
MESSAGE abc$
```

Displays "I want to dance" in a message box.

Combine LEFT with INSTR to extract the portion of a string either up to or including a specified substring.

```
city$ = "San Francisco, California"
Mystr$ = LEFT(city$, INSTR(city$, ",")-1)
```

Extracts the characters in **city$** up to, but not including the comma, and places the result in the variable **Mystr$**. The variable **Mystr$** now has the value "San Francisco".

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

LEN function

LEN(source)

Returns the length or number of characters in a string.

Parameter	Description
source	The string expression that is measured.

Example

```
num = LEN("This is a test")
```

Assigns the length of the string, 14, to the variable **num**.

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

LTRIM function

LTRIM(source)

Removes any leading spaces from a string. You can use **LTRIM** to remove leading spaces from dialog box inputs.

Parameter	Description
source	The string expression from which leading spaces are removed.

Example

```
MyString$ = "   Test"
MyString$ = LTRIM(MyString$)
```

Assigns "Test" to the variable **MyString$**. All leading spaces that were previously in the variable are removed.

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

MID function and statement

Function: MID(source, index, count**)**
Statement: MID(source, index , count**) = modify**

If used as a function, MID returns a specified number of characters, starting at a specified position in a string. If used as a statement, MID replaces a portion of a string with another string, beginning at a specified character.

Parameter	Description
source	Any string, string variable, string constant, or expression returning a string. Hold the string to be modified.
source	For a function, the string expression from which to return characters. For a statement, the string expression holding the original string to be modified.
index	Position of the first character to be returned (function) or modified (statement).
count	For a function, the number of characters to be returned. For a statement, the number of characters to be overwritten. If not specified, the rest of **source** is returned or overwritten.
modify	A string expression replacing a portion of **source**.

Note

- You can use the **LEN** function to determine the number of characters in a string.

Example

```
s$ = MID("I want to dance with you", 11, 5)
```

The function extracts five characters from the string, beginning with the eleventh character. The variable **s$** then becomes "dance".

```
str1$ = "I want to dance with you"
MID(str1$, 22, 3) = "him"
```

The statement changes three characters, starting with the 22nd character, to the new string. The **str1$** variable now contains "I want to dance with him".

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} **Related Topics**

RIGHT function

RIGHT (source, number)

Returns the specified number of characters from the end of a string.

Parameter	Description
source	The string expression from which the specified characters are returned.
number	Lets you specify the number of characters to be returned. Must be a non negative number and fractional numbers are rounded.

Note

- This function can be used to truncate user input from a dialog box.
- You can use the **LEN** function to determine the number of characters in a string.
- You can use the **LEFT** function to return characters from the beginning of a string.

Example

```
abc$ = RIGHT("I don't want to dance", 13)
MESSAGE abc$
```

Displays "want to dance" in a message dialog box.

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

RTRIM function

RTRIM(source)

Removes any trailing spaces from a string. You can use RTRIM to remove trailing spaces from dialog box inputs.

Parameter	Description
source	The string expression from which trailing spaces are removed.

Example

```
MyString$ = "Test   "
MyString$ = RTRIM(MyString$)
```

Assigns "Test" to the variable **MyString$**. All trailing spaces that were previously in the variable are removed.

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

SPACE function

SPACE (num)

Returns a string that consists of a specified number of spaces (ANSI character number 32).

Parameter	Description
num	Lets you specify the number of spaces to be included in a string.

Note

- See the Corel SCRIPT Character Map for more ANSI details and Windows characters.

Example

```
Mystr$ = SPACE(4) + "Test" + SPACE(4)
```

Makes the string variable **Mystr$** equal to " Test ". (The string **Mystr$** now consists of 4 spaces, the word TEST and another 4 spaces).

```
x1 = "Corel"
x2 = "SCRIPT"
x3 = x1 + SPACE(1) + x2
```

Makes the string variable **x3$** equal to "Corel SCRIPT".

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

STR function

STR(num)

Returns a string representation of a numeric data type. The **STR** function is useful when you want to manipulate a number as a string.

Parameter	Description
num	Lets you specify the numeric expression that is returned as string representation.

Note

- If a positive number is converted, the **STR** function inserts a leading space before the first character. If a negative number is converted, the **STR** function inserts a negative sign before the first character.
- You can use only the period as a decimal separator with the **STR** function. If you're not using a period (**.**) as a decimal separator, the **CSTR** function can be used to convert a number to a string. Your Windows decimal settings are set in the Control Panel.
- If you're converting dates, they must be in the standard International format (yy/MM/dd hh:mm:ss).

Example

```
aInteger$ = STR(72)
aNonInteger$ = STR(.140166)
```

The first example assigns "72" to the variable **aInteger$**. The second example assigns "0.140166" to **aNonInteger$**.

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

UCASE function

UCASE(source)

Converts a string to uppercase characters.

Parameter	Description
source	The string expression to convert.

Note

- Non letter characters do not change when this function is used.
- You can use **LCASE** to convert to lowercase characters.

Example

```
x$="hI"
firststring$ = UCASE (x)
secondstring$ = UCASE("THerE")
MESSAGE firststring + " " + secondstring
```

Displays the converted strings "HI THERE" in a message dialog box.

{button ,AL(`cs_strings_fns;;;;;',0,"Defaultoverview",)} Related Topics

VAL function

VAL (chars)

Converts a string to a number. The number's variable type is double. This function is the opposite of the **STR** function.

Parameter	Description
chars	The string expression to be converted. If the string does not begin with a number, VAL returns 0.

Note

- You can use only the period as a decimal separator with the VAL function. If you're not using a period (**.**) as a decimal separator, the **CDBL** or **CSNG** function can be used to convert a string to a number. Your Windows decimal settings are set in the Control Panel.
- The **VAL** function converts the string up to the first non-number character it encounters, from left to right in the string. Spaces are ignored.
- Because text box controls in dialog boxes can only return strings (even if numbers are input), you can use the **VAL** function to convert strings entered in dialog boxes to numbers.

Example

```
g = VAL("72nd Street")
h = VAL("72.700113")
```

Both the above statements assign 72 to the variables **g** and **h**.

{button ,AL(`cs_strings_fns;inputbox;;;;',0,"Defaultoverview",)} Related Topics

Mathematical functions

You can use to the following formulas to create mathematical functions in Corel SCRIPT scripts. The formulas are based on Corel SCRIPT functions.

Function	To use this...	...paste this formula into your script
Secant	SEC(X) =	1 / COS(X)
Cosecant	COSEC(X) =	1 / SIN(X)
Cotangent	COTAN(X) =	1 / TAN(X)
Arc secant	ARCSEC(X) =	ACOS (1 / X)
Arc cosecant	ARCCOSEC(X) =	ASIN (1 / X)
Arc cotangent	ARCCOTAN(X) =	ATAN (1 / X)
Hyperbolic sine	HSIN(X) =	(EXP(X)- EXP (-X))/ 2
Hyperbolic cosine	HCOS(X) =	(EXP(X)+ EXP (-X))/ 2
Hyperbolic tangent	HTAN(X) =	(EXP(X)- EXP (-X)) / (EXP(X)+ EXP (-X))
Hyperbolic secant	HSEC(X) =	2 / (EXP(X)+ EXP (-X))
Hyperbolic cosecant	HCOSEC(X) =	2 / (EXP(X)- EXP (-X))
Hyperbolic cotangent	HCOTAN(X) =	(EXP(X)+ EXP (-X)) / (EXP(X)- EXP (-X))
Inverse hyperbolic sine	HARCSIN(X) =	LOG (X + SQR (X * X + 1))
Inverse hyperbolic cosine	HARCCOS(X) =	LOG (X + SQR (X * X -1))
Inverse hyperbolic tangent	HARCTAN(X) =	LOG ((1 + X)/(1 - X)) / 2
Inverse hyperbolic secant	HARCSEC(X) =	LOG ((SQR (-X * X + 1) + 1) / X
Inverse hyperbolic cosecant	HARCCOSEC(X)	LOG ((SQR (X) * SQR (X * X + 1) +1) / X)
Inverse hyperbolic cotangent	HARCCOTAN(X) =	LOG ((X + 1)/(X - 1)) / 2

Note

- You can copy a formula into a script by selecting it, clicking right-mouse button, Copy. The formula is placed on the clipboard ready to be pasted.

{button ,AL(`all_math_fns;;;;;',0,"Defaultoverview",)} **Related Topics**

CBOL function

CBOL(NumStrExp)

Converts an expression to a Boolean data type. Corel SCRIPT data type conversion functions set the result of an expression to a specified data type rather than the default data type.

Parameter	Description
NumStrExp	A numeric or string expression.

Note

- A numeric expression equal to 0 is converted to FALSE. All other numeric expressions are converted to TRUE (-1).

Example

```
x% = 354.43
y = CBOL(x)
```

This example sets **y** to TRUE (-1).

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;;;;',0,"Defaultoverview",)} Related Topics

CCUR function

CCUR(NumStrExp)

Converts an expression to a Currency data type. Corel SCRIPT data type conversion functions set the result of an expression to a specified data type rather than the default data type.

Parameter	Description
NumStrExp	A numeric or string expression.

Notes

- When Corel SCRIPT converts the value to the currency data type, it rounds off the value, rather than truncating it.
- If the result of this function lies outside the Currency data type range, an error occurs.

Example

```
x = 354.432675434
y = CCUR(x*2)
```

This example sets **y** to a currency data type.

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;;;;',0,"Defaultoverview",)} Related Topics

CDAT function

CDAT(NumStrExp)

Converts an expression to a Date data type. Corel SCRIPT data type conversion functions set the result of an expression to a specified data type rather than the default data type.

Parameter	Description
NumStrExp	A numeric or string expression.

Notes

- This function denotes a base date of December 31, 1899 at 12:00:00 A.M. as 1. Each additional whole number is one additional day. Each additional fraction is a portion of a day.
- When Corel SCRIPT converts the value to a date, it rounds off the value, rather than truncating it.

Example

```
x% = 25.25
y = CDAT(x)
```

This example sets **y** to January 24th, 1900 at 6:00:00 A.M.

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;;;;',0,"Defaultoverview",)} Related Topics

CDBL function

CDBL(NumStrExp)

Converts an expression to a Double data type. Corel SCRIPT data type conversion functions set the result of an expression to a specified data type rather than the default data type.

Parameter	Description
NumStrExp	A numeric or string expression.

Notes

- When Corel SCRIPT converts the value to a double, it rounds off the value, rather than truncating it.
- If the result of this function lies outside the Double data type range, an error occurs.

Example

```
x@ = 35489097326.43    'x is currency data type
y = CDBL(x/2)
```

This example sets **y** to double data type.

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;;;;',0,"Defaultoverview",)} Related Topics

CINT function

CINT(NumStrExp)

Converts an expression to a Integer data type. Corel SCRIPT data type conversion functions set the result of an expression to a specified data type rather than the default data type.

Parameter	Description
NumStrExp	A numeric or string expression.

Notes

- In converting the value to an integer, Corel SCRIPT rounds off the value, rather than truncating it. If the decimal portion of the number is 0.5, CINT rounds to the nearest even number. For example, 8.5 rounds to 8, and 9.5 rounds to 10.
- If the result of this function lies outside the Integer data type range, an error occurs.
- You can also use INT and FIX to remove the fractional portion of a number.

Example

```
x = 354.63
y = CINT(x)
```

This example sets **y** to 355.

```
xx = 354.43
yy = CINT(xx)
```

This example sets **yy** to 354.

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;int;fix;;',0,"Defaultoverview",)} Related Topics

CLNG function

CLNG(NumStrExp)

Converts an expression to a Long data type. Corel SCRIPT data type conversion functions set the result of an expression to a specified data type rather than the default data type.

Parameter	Description
NumStrExp	A numeric or string expression.

Notes

- In converting the value to a long integer, Corel SCRIPT rounds off the value, rather than truncating it.
- If the result of this function lies outside the Long data type range, an error occurs.
- You can also use INT and FIX to remove the fractional portion of a number.

Example

```
x = 98765578.43
y = CLNG(x)
```

This example sets **y** to 98765578.

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;int;fix;;',0,"Defaultoverview",)} Related Topics

CONST statement

GLOBAL **CONST constant**{%|&|!|#|@|$} **= expression** , constant{%|&|!|#|@|$} = expression, ...
GLOBAL **CONST constant** AS type **= expression**, constant AS type = expression, ...

Declares constants for use in place of numeric or string values. Changing the value of a constant normally requires editing only one script statement.

Parameter	Description
GLOBAL	An optional parameter used to declare global constants. Global constants are available to all procedures in a script. If not used, the constant is only available to the procedure in which it was declared.
constant{%|&|!|#|@|$}	Lets you specify the name of the constant. It follows the Corel SCRIPT naming convention. Optionally, a type-declaration character can follow the name.
constant	Lets you specify the name of the constant. It follows the Corel SCRIPT naming convention.
AS type	Declares the data type of the constant with a type declaration name.
expression	A numeric or string expression assigned to the declared constant.

Notes

- You cannot declare a constant as a variant.
- Unlike variables, you can't change or assign a new value to a constant once it has been declared.
- Constants can be used to declare the size of an array.
- If you re-use many of the same constants, you should consider putting them in a separate script. Keeping this information in a separate script allows you to type the information once and then call it as many times as you need with an **INCLUDE** statement in any new script you create.

Example

```
REM creates a global constant for the base of the natural logarithm
GLOBAL CONST NATURAL_LOG# = 2.71828182845

REM creates a local constant for pi
CONST PI AS DOUBLE = 3.1415926535
```

{button ,AL(`script_procedures;global;dim;using_variables;Using_constants;corel_script_data_type_summary;;;',0,"Defaultoverview",)} Related Topics

CSNG function

CSNG(NumStrExp)

Converts an expression to a Single data type. Corel SCRIPT data type conversion functions set the result of an expression to a specified data type rather than the default data type.

Parameter	Description
NumStrExp	A numeric or string expression.

Notes

- In converting the value to a single, Corel SCRIPT rounds off the value, rather than truncating it.
- If the result of this function lies outside the Currency data type range, an error occurs.

Example

```
x = 354
y = CSNG(x + 0.02)
```

This example sets **y** to a single data type.

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;;;;',0,"Defaultoverview",)} Related Topics

CSTR function

CSTR(NumStrExp)

Converts an expression to a String data type. Corel SCRIPT data type conversion functions set the result of an expression to a specified data type rather than the default data type.

Parameter	Description
NumStrExp	A numeric or string expression.

Notes

- A Boolean value of 0, when cast as a string, will return "FALSE". All other values will return "TRUE".
- Converting a date returns a string with a date in the Windows long-date format.
- You can also use **STR** to convert numeric expressions to a string.
- The **CSTR** function is locale aware; that is, it uses the decimal character specified in the user system's regional settings. In Windows 95, click Start, Setting, Control Panel, Regional settings to view and change your system settings.

Example

```
x = 354.43
y = CSTR(x)
```

This example sets **y** to "354.43", a string.

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;STR;;;',0,"Defaultoverview",)} Related Topics

DEFINE statement

#DEFINE substituteID syntax : syntax: ...

This statement is used to create a substitute for Corel SCRIPT syntax. Using a substitute is similar to using a constant. A substitute can be used to provide meaningful names for numeric and string values but it can also be used to replace script statements and functions.

Parameter	Description
substituteID	Lets you specify the identifier for the substituted Corel SCRIPT syntax.
syntax : syntax: ...	Lets you specify the Corel SCRIPT to substitute.

Notes

- You can remove a defined substitute from a script by using the **UNDEF** statement.
- The number sign (#) is required in the syntax.

Example

```
#DEFINE ErrorBox MESSAGE "An error occurred" : MESSAGE "FIX IT!"
```

The above example replaces two MESSAGE statements with the substitute **ErrorBox**.

{button ,AL(`const;global;define;undef;;',0,"Defaultoverview",)} Related Topics

DIM statement

Syntax for variables:
DIM variable{%|&|!|#|@|$}, variable AS type,...

Syntax for arrays:
DIM array_name{%|&|!|#|@|$} **(upperbound)**
DIM array_name(upperbound) AS type
DIM array_name{%|&|!|#|@|$} **(lowerbound TO upperbound)**
DIM array_name(lowerbound TO upperbound) AS type

Use **DIM** to declare local variables explicitly or to specify the number and type of elements in an array. Local variables are available only to one procedure in a script.

Parameter	Description
variable{%|&|!|#|@|$}	
or **variable AS type**	Lets you specify the name of the variable and follows the Corel SCRIPT naming convention. The variable can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option).
array_name {%|&|!|#|@|$}	
or **array_name AS type**	Lets you specify the name of the array and follows the Corel SCRIPT naming convention. The array can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option).
upperbound	The upper bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **upperbound**) is used.
lowerbound	The lower bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **upperbound**) is used.

Note

- If you declare a variable or array using the **DIM** statement without a type declaration name or a type-declaration character, Corel SCRIPT sets the variable or array to a variant data type.
- Variables and arrays declared in the main section of a script are available in the main section only. Global variables are available to all procedures in a script. See **GLOBAL** for more information.
- Declaring a variable in a subroutine or a function makes it available only in the procedure it was declared. See Variable availability for more information about using variables in procedures.
- It's a generally accepted programming convention to put declaration statements at the beginning of a procedure (main section, subroutines, or functions).
- The **DIM** statement can be placed anywhere in a script before the variable(s) it declares is called.
- Arrays can hold only one data type. The number of elements arrays can hold is limited to your system's memory.
- See Multi-dimensional arrays to create arrays of more than one dimension.
- You can re-dimension (or resize) arrays using the **REDIM** statement.

{button ,AL(`Explicitly_declaring;dim;redim;global;using_arrays;using_variables;lbound;ubound;multi_dimensional_arrays;',0,"Defaultoverview",)} Related Topics

GETTYPE function

GETTYPE(expression)

Returns an expression's data type. In the case of variants, the data subtype is returned. See Variants for more information.

Return Value

The GETTYPE function returns one of the following values:

- 0⇨Empty variant
- 2⇨Boolean
- 3⇨Integer
- 4⇨Long
- 5⇨Single
- 6⇨Double
- 7⇨Date
- 8⇨Currency
- 9⇨String

Parameter	Description
expression	Lets you specify the expression to use.

Example

```
x% = 66    'integer data type
y# = 14    'long data type
Z = x / y
A = GETTYPE (x / y)
```

In the above example, **A** is assigned the value 2 since the variant **Z** data subtype is set to long. In the following example, **B** is set to 4 since a whole number is treated as a long and **C** is set to 6 since a fractional number is treated as a double.

```
B = GETTYPE (3)
C = GETTYPE (3.3)
```

{button ,AL(`all_vars;dim;;;;',0,"Defaultoverview",)} Related Topics

GLOBAL statement

Syntax for variables
GLOBAL variable{%|&|!|#|@|$}, variable AS type,...

Syntax for arrays
GLOBAL array_name{%|&|!|#|@|$} **(upperbound)**
GLOBAL array_name(upperbound) AS type
GLOBAL array_name{%|&|!|#|@|$} **(lowerbound TO upperbound)**
GLOBAL array_name(lowerbound TO upperbound) AS type

Use GLOBAL to explicitly declare variables or to specify the number and type of elements in an array. Global variables and arrays are available to all procedures in a script. See the **DIM** statement for information on local variable declarations. See Variable availability for more information on the use of variables in procedures.

Parameter	Description
variable{%|&|!|#|@|$}	
or **variable AS type**	Lets you specify the name of the variable and follows the Corel SCRIPT naming convention. The variable can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option).
array_name {%|&|!|#|@|$}	
or **array_name AS type**	Lets you specify the name of the array and follows the Corel SCRIPT naming convention. The array can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option).
upperbound	The upper bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **upperbound**) is used.
lowerbound	The lower bound of the array expressed as an integer.

Notes

- If you declare a variable or array using the **GLOBAL** statement without a type declaration name or a type-declaration character, Corel SCRIPT sets the variable or array to a variant data type.
- Global variables cannot be declared in a subroutine or a function. Additionally, globals cannot be declared in a flow construct such as **FOR...NEXT** or **DO...LOOP**. This restriction also applies to arrays.
- It's a generally accepted programming convention to put declaration statements at the beginning of the script.
- Arrays can hold only one data type. The number of elements arrays can hold is limited by your system memory.
- You can also declare global constants. See **CONST** for more information.
- See Multi-dimensional arrays to create arrays of more than one dimension.

{button ,AL(`const;dim;global;using_arrays;using_variables;lbound;ubound;multi_dimensional_arrays;',0,"Defaultoverview",)} Related Topics

Examples for DIM statement

Variables

```
DIM my_color$
DIM my_color AS STRING
```

The above examples show different methods of declaring variables. The above DIM statements all declare strings.

```
DIM a AS INTEGER, b AS BOOLEAN, c AS SINGLE
```

You can also mix the type of variables you declare with a DIM statement.

Arrays

```
DIM color$(5)
color$(1) = "black"
color$(2) = "red"
color$(3) = "white"
color$(4) = "blue"
color$(5) = "green"
```

Creates a string array named **color$** that consists of 5 elements.

```
DIM salespeople(-2 TO +3) AS INTEGER
salespeople(-2) = 1
salespeople(-1) = 3
salespeople(0) = 5
salespeople(1) = 7
salespeople(2) = 9
salespeople(3) = 11
```

Creates an integer array named salespeople that consists of 6 elements.

Note

- See Multi-dimensional arrays to create arrays of more than one dimension.

{button ,AL(`example_multi_array;dim;global;using_variables;using_arrays;',0,"Defaultoverview",)} Related Topics

Examples for GLOBAL statement

Variables

```
GLOBAL my_color$
GLOBAL my_color AS STRING
```

The above examples show different methods of declaring global variables. The above GLOBAL statements all declare strings.

```
GLOBAL a AS INTEGER, b AS BOOLEAN, c AS SINGLE
```

You can also mix the type of variables you declare with a GLOBAL statement.

Arrays

```
GLOBAL color$(5)
color$(1) = "black"
color$(2) = "red"
color$(3) = "white"
color$(4) = "blue"
color$(5) = "green"
```

Creates a global string array named **color$** that consists of 5 elements.

```
GLOBAL salespeople(-2 TO +3) AS INTEGER
salespeople(-2) = 1
salespeople(-1) = 3
salespeople(0) = 5
salespeople(1) = 7
salespeople(2) = 9
salespeople(3) = 11
```

Creates an global integer array named salespeople that consists of 6 elements.

Note

- See Multi-dimensional arrays to create arrays of more than one dimension.

{button ,AL(`example_multi_array;dim;global;using_variables;using_arrays;',0,"Defaultoverview",)} Related Topics

LBOUND function

LBOUND(array, dimension**)**

Returns the lower bound for a specified dimension of an array.

Parameter	Description
array	Lets you specify the array to dimension.
dimension	A whole number variable or numeric constant ranging from 1 to the number of dimensions in the array. Lets you specify which dimension's lower bound is returned. If omitted, the limit of the first dimension is returned.

Example

```
DIM a%(-5 TO 7, 10)
x% = LBOUND(a%,1)
y% = LBOUND(a%,2)
```

Sets **x%** and **y%** to -5 and 1, respectively.

{button ,AL(`LBOUND;UBOUND;DIM;USING_ARRAYS;multi_dimensional_arrays;',0,"Defaultoverview",)} **Related Topics**

LET statement

LET **variable**{%|&|!|#|@|$} **= expression**
LET **variable = expression** AS type
variable{%|&|!|#|@|$} **= expression**
variable = expression AS type

Assigns the value of an expression to a variable. The **LET** keyword is optional.

Parameter	Description
variable{%|&|!|#|@|$}	Lets you specify the name of variable and is assigned **expression**'s value. The variable name follows the Corel SCRIPT naming convention.
variable	Lets you specify the name of variable and is assigned **expression**'s value. The variable name follows the Corel SCRIPT naming convention.
expression	A numeric or string expression that is assigned to the variable.
type	Declares the variable's type with a type declaration name.

Note

- There isn't an advantage in using the LET statement to assign an expression to a variable, but in some cases it can make your script easier to read and modify.
- If the variable's data type is not declared, the variable is set to the variant data type.

Example

```
LET stringVar$ = "This is a string."
```

Assigns the string "This is a string." to the variable **stringVar$**.

```
stringVar$ = "This is a string."
```

Assigns the string "This is a string." to the variable **stringVar$**. The LET keyword is omitted.

```
result% = (a% + b%) / c%
```

Assigns the result of the sum of the values of variables **a**% and **b%**, divided by the value of **c%**, to the variable **result%**. The LET keyword is omitted.

{button ,AL(`variable_availability;using_variables;Dim;;;',0,"Defaultoverview",)} Related Topics

REDIM statement

REDIM PRESERVE **array_name**{%|&|!|#|@|$} **(upperbound)**
REDIM PRESERVE **array_name(upperbound)** AS type
REDIM PRESERVE **array_name**{%|&|!|#|@|$} **(lowerbound TO upperbound)**
REDIM PRESERVE **array_name(lowerbound TO upperbound)** AS type

Used to re-dimension (change the number of array elements or dimensions) in a previously declared array. See the **DIM** statement for information about declaring arrays.

Parameter	Description
array_name {%|&|!|#|@|$}	
or **array_name AS type**	Lets you specify the name of the array and follows the Corel SCRIPT naming convention. The array can be declared using a type-declaration character (the first option)or by using a type declaration name (the second option).
upperbound	The upper bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **upperbound**) is used.
lowerbound	The lower bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **upperbound**) is used.
PRESERVE	Corel SCRIPT keyword which when used indicates to preserve the data in the array that is being re-dimensioned. Use the PRESERVE keyword to retain data in the elements that are part of the re-dimensioned array. If this keyword is not used, all the array data is discarded.

Note

- You can't use the **REDIM** statement to change the variable type an array can hold.

Example

```
DIM color$(5)
color$(1) = "black"
color$(2) = "red"
color$(3) = "white"
color$(4) = "blue"
color$(5) = "green"
```

The above example creates a string array with 5 elements. The following scripting command removes the last two elements from the **color** string array by using the REDIM statement.

```
REDIM PRESERVE color$(3)
```

{button ,AL(`dim;redim;global;using_arrays;using_variables;lbound;ubound;multi_dimensional_arrays;',0,"Defaultoverview",)} Related Topics

SETEMPTY statement

SETEMPTY variable

This statement removes data from a variable. If the variable is numeric, the variable is set to 0. If the variable is a string, the variable is set to an empty string (""). If the variable is a variant, the variable is set to Empty. An Empty variant contains no valid data in both a numeric and string context. See Corel SCRIPT data type summary for more information about data types and variants.

Parameter	Description
variable	Lets you specify the name of the variable to remove data from.

Example

```
SETEMPTY VariableName
```

The above example remove data from the **VariableName** variable.

{button ,AL(`variants;variable_availability;using_variables;Dim;;;',0,"Defaultoverview",)} Related Topics

STATIC statement

STATIC variable{%|&|!|#|@|$} = expression
STATIC variable AS type = expression

This statement declares and assigns an initial value to a variable in a user-defined subroutine or function. Static variables can only be called in these user-defined procedures, and retain their values as long as the script they are declared in is running. See Variable availability for more information about using variables in procedures.

Parameter	Description
variable{%|&|!|#|@|$}	Lets you specify the name of the variable to be declared. The variable name follows the Corel SCRIPT naming convention.
variable	Lets you specify the name of the variable to be declared. The variable name follows the Corel SCRIPT naming convention.
type	Declares the variable's data type with a type declaration name.
expression	The numeric expression initially assigned to the declared static variable. This option cannot be used with static variables of string or variant data types. Declaring and assigning a value to constant at the same time is new to Corel SCRIPT in version 7.0.

Note

- If you declare a variable using the **STATIC** statement without a type declaration name or a type-declaration character, Corel SCRIPT sets the variable to a variant data type.
- It's a generally accepted programming convention to put static declaration statements at the beginning of subroutines or functions with **DIM** declarations.

{button ,AL(`function_end_function;sub_end_sub;variable_availability;using_variables;Dim;;;',0,"Defaultoverview",)}
Related Topics

Example for STATIC statement

```
REM main section of script file
DECLARE FUNCTION staticFunc% (a%)
FOR i% = 1 to 5
    j% = staticFunc%(i%)
NEXT I%
'
REM (Static Function Example)
FUNCTION staticFunc%(a%)
STATIC staticVar%
' Because staticVar% is STATIC, it retains its previous value
' each time the function is called
staticVar% = staticVar% + a%
' The function returns the current value of staticVar%
staticFunc% = staticVar%
END SUB
```

The variable **staticVar%** in the function is created as a STATIC variable, so that its value remains unchanged each time the function is called. In the main program, a FOR loop calls the function five times. The result of each function call follows:

1 The first time the script runs, **staticVar%** has a value of 0 because it is created for the first time. The passed parameter, **i%**, has a value of 1, and the variable also has a value of 1.

2 In the second call, **staticVar%** has a value of 1 and the passed parameter has a value of 2. So the calculation causes **staticVar%** to be 3.

3 In the third call, **staticVar%** is equal to 3 and **i%** is equal to 3, so **staticVar%** has a new value of 6.

4 In the fourth call, **staticVar%** is 6 and **i%** is 4, giving **staticVar%** a new value of 10.

5 In the last call, **staticVar%** is 10 and **i%** is 5, giving **staticVar%** a value of 15.

{button ,AL(`example_vars;;;;;',0,"Defaultoverview",)} Related Topics

UBOUND function

UBOUND(array, dimension**)**

Returns the upper bound for a specified dimension of an array.

Parameter	Description
array	Lets you specify the array to dimension.
dimension	A whole number variable or numeric constant ranging from 1 to the number of dimensions in the array. Lets you specify which dimension's upper bound is returned. If omitted, the limit of the first dimension is returned.

Example

```
DIM a%(-5 TO 7, 10)
x% = UBOUND(a%,1)
y% = UBOUND(a%,2)
```

Sets **x%** and **y%** to 7 and 10, respectively.

{button ,AL(`redim;LBOUND;UBOUND;DIM;USING_ARRAYS;multi_dimensional_arrays;',0,"Defaultoverview",)} Related Topics

UNDEF statement

#UNDEF substituteID

This statement is used to remove a substitute declared with a **DEFINE** statement from a script.

Parameter	Description
substituteID	Lets you specify the identifier of the substituted Corel SCRIPT syntax to remove.

Note

- The number sign (#) is required in the syntax.

Example

```
#UNDEF ErrorBox
```

The above example removes the substitute **ErrorBox**.

{button ,AL(`const;global;define;undef;;',0,"Defaultoverview",)} **Related Topics**