Chapter 10 The Parser

Any mindmap component may contain a formula that produces a result. A formula may be associated with a component or embedded in the component. In order to embed (you), you invoke the Formula Editor and define the formula. If you wish to associate a formula with an object, this is accomplished in any of the many Link dialog boxes. Input fields that are capable of dealing with formulas can be recognized by a small command button in their upper right corner which displays a question mark.

You may create formulas using:

- constants or values
- mathematical, comparison, and logical operators
- references to other components
- built-in routines called functions

You may combine these elements into an expression. For example, a formula for a field 'Invoice Total' could be:

```
(Subtotal * 1.12) + Shipping
```

Overview

Constants

Constants are values in formulas that do not change:

<u>Examples</u>:

- `Shipping * 1.12` ; 1.12 is a constant
- `"Due " + date` ; 'Due' is a text constant

When you include a text constant in a formula, you must enclose the text constant with either single or double quotation marks. To include quotation marks within a text constant, you must use backslash quotation marks:

Example:

- `"Date \"Second Notice\" Due: "`
  ; results in Date 'Second Notice' Due:

Operators

Operators are symbols indicating how to combine two or more expressions. They include the standard arithmetic operators (+, -, *, /, ^, and %) and some special operators discussed later:

- `SubTotal * 1.12` ; * is the multiplication operator
- `"Date due "+ strdate`
  ; + combines text values

Component References

Component names may be used in formulas to include the contents of the component in the formula. When mindmap evaluates the formula, the contents of the component replaces the name of the component.

Examples:

- `SubTotal * 1.12` ; SubTotal is a component name
- `SubTotal / AmountOrders`
  ; AmountOrders is a component

Functions

Functions perform frequently used calculations (or operations) and are generally followed by parameters (values or references to components). Functions are described in detail later.

Examples:

- `date` ; Supplies the current system date
- `cos(edt1)` ; returns the cosine value of edt1

Expressions

An expression is a logical sequence of functions, values, constants, and operators working together to return a single result. Expressions are like clauses and phrases in a sentence.

Examples:

- `(SubTotal * 1.12) + Shipping`
- `substr(Title, 2, 5) + Name`

Using appropriate operators and functions, you may construct various kinds of values within a formula. However, a formula may have only one type of result:

- string
- numeric
- date
- time

The internal representation of strings is ASCIIZ. Call-by-Name references to strings are guaranteed to supply a buffer size of 4096 bytes, including the terminating null character. Numeric values are represented as doubles. Dates are represented as unsigned long (32-bit) values, interpreted as Julian date. Time is an unsigned long (32-bit) value, interpreted as seconds.

You must take care in constructing expressions with regards to the various types of data used. Using incorrect types might lead to unexpected results. For numerical errors occurring in mathematical functions of the parser, the value expected to be returned from the function is replaced with 0, if the function reports an error. An error message is appended to the system log file, as well as to the parser window, if applicable (see mindmap menu **File | Preferences | Formula** on page 46).

Currently, the math runtime library supports the following error messages:

- 'Numerical overflow in function fn(arg)'
- 'Invalid argument range in function fn(arg)'
- 'Argument singularity in function fn(arg)'
- 'Partial loss of precision in function fn(arg)'

- 'Total loss of precision in function fn(arg)'

where fn(arg) is replaced with the function name and argument that caused the error.

Please note, that for object-registered functions, the parser only defines the requested data type for passed arguments, whereas the data type returned by the function may depend on the specific implementation. For example, the function GetProp, registered by the VBX object, returns a data type depending on the name of the property. Passing this value directly to a function might cause problems, if the function expects a different type. These problems have been avoided by having mindmap automatically perform a reasonable conversion before passing the argument.

mindmap automatically converts into appropriate types and appends a warning to the system log. This applies also to operators that combine different types. Almost always, mindmap attempts to concatenate the factors as strings.

Operators

mindmap supplies various types of operators to work with expressions:

- mathematical
- comparison
- logical
- text

By definition, operators are placed between two expression elements. The following tables show all the mindmap operators and indicate how two values are combined:

Mathematical Operators

Symbol	Name	Definition and Example
+	Plus	Add two values `2 + 2`

		`SubTotal + Shipping`

-	Minus	Subtracts second value from first value
		`2 - 2`
		`SubTotal - Discount`

*	Multiply	Multiplies two values
		`2 * 2`
		`SubTotal * SalesTax`

/	Divide	Divides the first value by the second
		`Meters/100`
		`500/2.5`
		An error message is generated if the divisor is zero.

Mod
%
	Modulus	Calculates the remainder of a division
		`5 mod 3` ;equals 2
		`6 mod 3` ;equals 0
		The percent symbol can be used in place of 'mod'.
		`5 % 3`
		`6 % 3`

^	Power	`10 ^ 2` ; equals 100 `2 ^ 5` ; equals 32 The result of this operation is 1 if both expressions are zero.
( )	Precedence	mindmap evaluates formulas from left to right, performing multiplications and divisions before additions and subtractions. Using parentheses allows you to change the order. mindmap evaluates expressions between parentheses first. `3 + 5 * 5` ;equals 28 `(3 + 5) * 5` ;equals 40

Comparison Operators

These operators compare two values and return either true or false. These expressions are often referred to as Boolean expressions. Arithmetically, a result of true equals 1 and a false equals 0.

Symbol	Name	Definition and Example
= ==	Equals	True if both elements are equal `23 = 29` ; false `27 = 27` ; true
<> !=	Not equals	True if both elements are not equal `23 <> 29` ; true `27 <> 27` ; false
>	Greater than	True if the value on the left of the operator exceeds the one on the right `23 > 29` ; false `23 > 23` ; false `23 > 19` ; true
<	Less Than	True if the value on the left of the operator is less than the value on the right `23 < 29` ; true `23 < 23` ; false `23 < 19` ; false
>=	Greater Than or Equal To	True if the value on the left is greater than or equal to the value on the right `23 >= 19` ; true `23 >= 23` ; true `23 >= 29` ; false
<=	Less Than or Equal to	True if the value on the left is less than or equal to the value on the right

		`23 <= 19` ; false `23 <= 23` ; true `23 <= 29` ; true

Logical Operators

Logical operators can build compound conditions into a formula. Sometimes, two or more conditions must be met before you choose a particular method of calculation. Logical operators enable you to describe such combinations of conditions.

Symbol	Definition and Example
AND	True only if both elements are true ‘true’ AND ‘true’ equals ‘true’ ‘true’ AND ‘false’ equals ‘false’ ‘false’ AND ‘true’ equals ‘false’ ‘false’ AND ‘false’ equals ‘false’
OR	True if either of the two values is true ‘true’ OR ‘true’ equals ‘true’ ‘true’ OR ‘false’ equals ‘true’ ‘false” OR ‘true’ equals ‘true’ ‘false’ OR ‘false’ equals ‘false’
XOR	True if either, but not both values, are true ‘true’ OR ‘true’ equals ‘false’ ‘true’ OR ‘false’ equals ‘true’ ‘false’ OR ‘true’ equals ‘true’ ‘false’ OR ‘false’ equals ‘false’
NOT	Changes the value of the subsequent Boolean operation from true to false or from false to true NOT 5 = 3 equals ‘true’

AND and OR are used to combine two expressions. NOT is only used on a single expression.

The result of a logical operation is treated as a numeric value having the possible values 0 or 1. Be aware that calculations with the results of logical operations are possible.

Text Operators

You may use text operators to indicate text constants in your formula or to combine two or more expressions into one expression.

Symbol	Name	Definition and Examples
+	Concatenate	Appends the text string on the right to the end of the text string on the left `"abc" + "def"` ; equals 'abcdef'

Special Symbols to be used in text constants

The following symbols are valid only within string constants (such as text enclosed in single or double quotation marks) and may be used to denote special characters in the text string.

Symbol	Name	Definition and Examples
" " ' '	Text Constant	Marks the beginning and the end of characters to be considered a text constant. Quotes without text between them indicate a blank space. If you enter text into a formula without using quotes, mindmap interprets the text as a component name or as a function.

		"edt1" ; a text string containing the four characters edt1 "mindmap" ; a text string containing the seven characters mindmap edt1 ; refers to a component with the name edt1
\\	Backslash	To represent a backslash in a text string, one has to put a backslash in front of it. 'c:\\mindmap\\samples' results in c:\mindmap\samples
\'	Double Quotation mark	To represent double quotation marks in a text string, a backslash has to be set in front of each quote. "Tool \"mindmap\"" results in Tool 'mindmap' Alternatively, the following form may be used: "Tool 'mindmap'" results in Tool 'mindmap'
\'	Single Quotation mark	To represent single quotation marks in a text string, a backslash has to be set in front of each quote. "Tool \"mindmap\"" results in Tool 'mindmap' Alternatively, the following form may be used: "Tool "mindmap"" results in Tool 'mindmap'
\n	New Line Character	To represent the new-line-character 0x0A. (This corresponds to the character code according to the ASCII table.)

\r	Carriage Return Character	To represent the carriage-return-character 0x0D. (This corresponds to the character code according to the ASCII table.)
\t	Tab Character	To represent the tab-character 0x09. (This corresponds to the character code according to the ASCII table.)

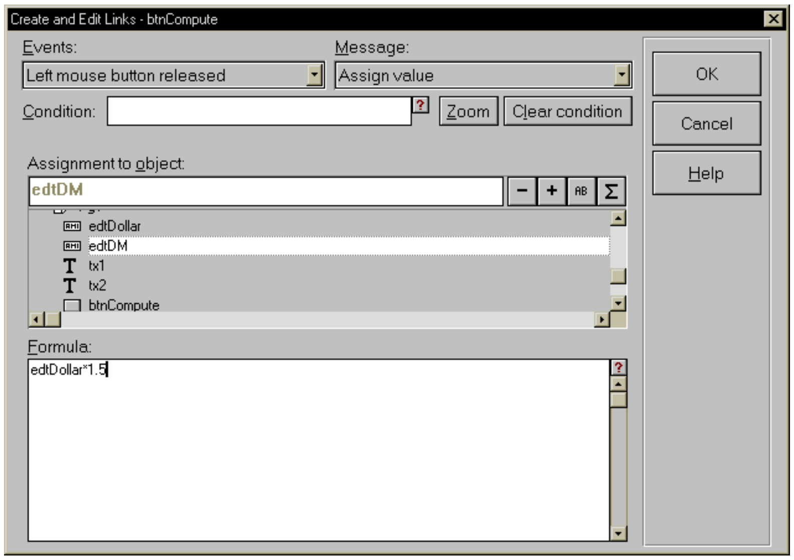
Formulas and functions

mindmap allows for any component to have a value, even if it is not apparent (i.e. command buttons) that this is the case. Some of the components are able to display their value. These include, for example, text and input fields. Other components cannot visualize their value, as is the case for graphical primitives, imported graphics, etc. In either case, it is possible to query the component for its value. A component may also include a formula. The result of the formula is the value of the component.

The values may be of the type numeric, string, or date. Strings are always enclosed in quotation marks. If a string is entered without the quotes, the parser will generate an error message (see ‘Parser Error Messages’ on page 391). Date constants are also enclosed in quotation marks, but they must follow the national rule of date formatting. Therefore, entering a calendar date depends on the currently selected language DLL (currently either MMDEU.DLL or MMENG.DLL).

Independent of the currently selected language, decimals in numeric constants are always separated by dot (not by comma). This convention makes a mindmap application independent of the language it was built in. However, input fields display numeric values (as well as dates) nationalized.

Assigning a Value via a Link

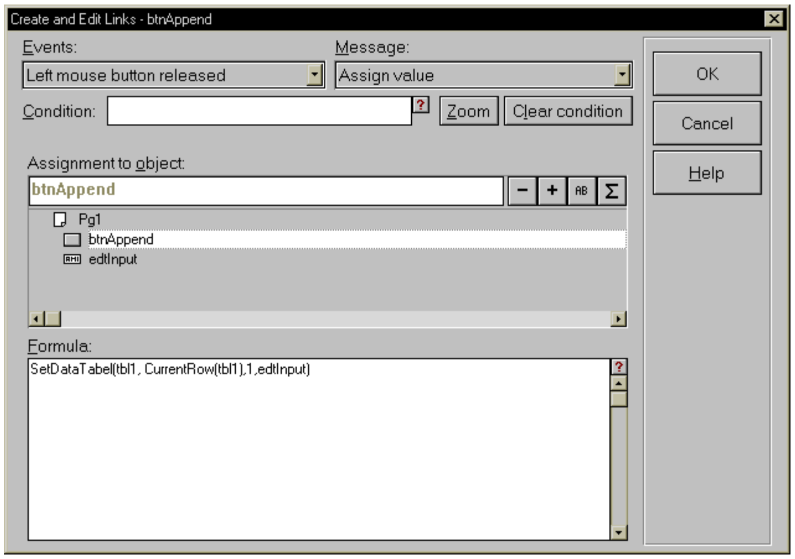


This link dialog box is used to assign a computed value to a component

This link, placed on the OK button, assigns the result of edt-Dollar multiplied by 1.5 to the input field edtDM.

Functions with an unspecified return value

In contrast to most formulas and functions, mindmap supports various functions which do not return a specific result. These functions are also used by assigning a value. In this case, you assign the value to a component which does not react to value assignments (i.e. command buttons). In such a case, the function is executed without changing the values of components.



Events: Left mouse button released

Message: Assign value

Assignment to object: btnAppend

Formula:

```
SetDataTabel(tbl1, CurrentRow(tbl1),1,edtInput)
```

mindmap will evaluate the function and ignore the result

In this example, the first column in the selected row in the data table tbl1 receives the value contained in edtInput. The function SetDataTable does not return a value.

➤ More information about assigning values to a component can be found on page 258.

Functions

A function is a built-in routine used to perform a specific calculation. Instead of typing what might be a complex formula, you simply type the function name. mindmap performs the calculations defined by the function.

You include the function name in your own calculation formulas, followed by the values you want the function to use. The information contained in the parentheses are called the parameters (also known as arguments). Most functions have one or more parameters that you must supply in order for mindmap to calculate the results you want.

Almost all mindmap functions contain these three basic parts:

- function name
- a set of parentheses
- the required parameters.

The function name tells mindmap what kind of work to do with the parameters you supply. The parentheses identify where the list of parameters begins and ends. When a function requires more than one parameter, the parameters must be separated with commas.

```
height(Rectangle1)
```

You use functions in your formulas by combining them with component names, operators, values, and other functions. Capitalization is not important when typing functions.

Function parameters may be constants, component names, expressions, or other functions. You may nest one function within another to perform more complex calculations with your data.

mindmap automatically converts arguments passed to functions into the required data type. This applies to either built-in, object-registered or functions imported through MNC files. If a function does not require arguments you should not supply parentheses.

<u>Example:</u>

`substr(datestr(date), 4,2)`
; extracts month from date

`substr(date,4,2)`
; the result will be the same because mindmap automatically converts the date to string since this is the required data type for the function substr.

General Functions

The following functions are always available. They are either intrinsic to mindmap or have been declared externally in the MMPARSE.MNC.

➤ For further details pertaining to the declaration of functions, please refer to page 479.

abs

Syntax:	abs(<num>)
Description:	Returns the absolute value of a number. If the number is negative, the abs function returns a positive value.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	The returned number is of the same data type as the parameter <num>.
Example:	`abs(edt1)` → `10` ; where edt1 contains the number 10 `abs(edt2)` → `10` ; where edt2 contains the number -10
See Also:	sign
Category:	Intrinsic function.

ANSI

Syntax:	ANSI(<text1>, <text2>, <num>)
Description:	Copies the first string into the second one, translating from the IBM-8 character set into the ANSI character set. The numerical value specifies how many characters are to be translated. The len function maybe used to calculate the number of characters available in the source string.
Parameter(s):	<text> is a mindmap component that can receive a text string. The <num > parameter is an integer.

Return Value(s):	A string is returned.
Example:	This function is especially used to translate language dependent special characters like German umlaut characters to the windows character set. `ANSI(edt1,edt2,5)` → "abcde" ; where edt2 contains "abcdefg"
See Also:	len
Category:	Declared function.

AppName

Syntax:	AppName (<component name>)
Description:	This function returns a string containing the full path name of the application that contains the given component.
Parameter(s):	name of a component
Return Value(s):	A string
Example:	`AppName (StartButton1)` → `C:\MINDMAP\APPS\LOGON.MM`
Category:	MMLIB

arccos

Syntax:	arccos(<num>)
Description:	Returns the arccosine of a number.
Parameter(s):	The <num> parameter is any expression that yields a number in the range -1 to 1. This is a value in radians.

Return Value(s):	The returned value is in radians. The formula for converting degrees to radians is radians=degrees*(/ 180)
Example:	`arccos(edt1) → 1.2238` ; where edt1 contains .34
See Also:	arctan, arcsin, sin, cos, tan
Category:	Intrinsic function.

arccosh

Syntax:	arccosh(<num>)
Description:	Returns the hyperbolic arccosine of an angle.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	The return value is numeric.
See Also:	sinh, cosh, tanh, arcsinh, arctanh
Category:	Intrinsic function.

arcsin

Syntax:	arcsin(<num>)
Description:	Returns the arcsine of a number.
Parameter(s):	The <num> parameter is any expression that yields a number in the range -1 to 1.
Return Value(s):	The returned value is in radians. The formula for converting degrees to radians is radians=degrees*(/ 180)
Example:	`arcsin(edt1) → .346916` ; where edt1 contains .34

See Also:	arctan, arccos, sin, cos, tan
Category:	Intrinsic function.

arcsinh

Syntax:	arcsinh(<num>)
Description:	Returns the hyperbolic arcsine of an angle.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	The return value is numeric.
See Also:	sinh, cosh, tanh, arccosh, arctanh
Category:	Intrinsic function.

arctan

Syntax:	arctan(<num>)
Description:	Returns the arctangent of a number.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	The returned value is in radians. The formula for converting degrees to radians is radians=degrees*(/ 180)
Example:	`arctan(edt1)` → `1.5374753` ; where edt1 contains 30.
See Also:	arcsin, arccos, sin, cos, tan
Category:	Intrinsic Function.

arctanh

Syntax:	arctanh(<num>)
Description:	Returns the hyperbolic arctangens of an angle.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	The return value is numeric.
See Also:	sinh, cosh, tanh, arcsinh, arccosh
Category:	Intrinsic function.

calc

Syntax:	calc(<text>)
Description:	This function evaluates the statement contained in the text string that is supplied as parameter.
Parameter(s):	The <text> parameter is a string representing a valid parser statement.
Return Value(s):	The outcome of this function depends on the statement. It has the same type that the statement evaluates to.
Example:	If the input field edt1 contains the value of 5 and the input field edt2 contains the text “edt1”, then the function `calc("7 * " + edt2)` ; will return 35.
Category:	Intrinsic function.

color

Syntax:	color (<num>,<num>,<num>)
Description:	This function converts three color values into a 32-bit RGB color value. The return value is calculated using the formula R*256*256+G*256+B The lower 8 bits represent the color value for blue, the next 8 bits represent the color value for green and the next 8 bits represent the color value for blue.

32..25	24..17	16..9	8..1
unused	red	green	blue

Parameter(s):	The three parameters specify the three color values for R, G and B respectively.
Return Value(s):	32-bit color value.
Example:	`color (255, 255, 255)` → `0x00ffffff` (hexadecimal for white)
See Also:	
Category:	MMLIB

CopyFile

Syntax:	CopyFile(<text>, <text>)
Description:	Copies the second file onto the first file. If the first file already exists, it will be overwritten.

Parameter(s):	The <text> parameter contains a valid file name with the path if the file is not in the current directory..
Return Value(s):	0 ; OK 1 ; Out of memory 2 ; Source file not found 3 ; Cannot create target file 4 ; Disc full 5 ; Wildcard file copy not successful

Example:	`CopyFile(edt1, edt2)`
	; where edt1 contains the text string "C:\FILE1.TXT" and edt2 contains the text string "FILE2.TXT"
	or
	`CopyFile("c:\\mm", "c:\\mindmap\\*.*")`

See Also:	DeleteFile
Category:	Declared function.

COS

Syntax:	cos(<num>)
Description:	Returns the cosine of a number.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	The returned value is in radians. The formula for converting degrees to radians is radians=degrees*(/ 180)
Example:	`cos(edt1) → .154251` ; where edt1 contains 30
See Also:	arccos, arcsin, arctan, tan, sin
Category:	Intrinsic function.

cosh

Syntax:	cosh(<num>)
Description:	Returns the hyperbolic cosine of an angle.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	The return value is numeric.
See Also:	sinh, tanh, arcsinh, arccosh, arctanh
Category:	Intrinsic function.

crlf

Syntax:	crlf
Description:	Inserts a carriage return/line feed.
Parameter(s):	none
Return Value(s):	Returns a string containing the carriage return and line feed characters.
Example:	An assign value command to an input field supplies the following results:

```
"Joe "+"Jones"
```
; Joe Jones

```
"Joe "+crlf+"Jones"
```
; Joe Jones

Category:	Intrinsic function

date

Syntax:	date

Description:	Supplies the current system date.
Parameter(s):	none.
Return Value(s):	A date in the format set in the MINDMAP.INI file which is dependent on what language (MMDEU.DLL or MMENG.DLL) is selected. Please note that you may set the date format in the application, by using the mask attribute. Then, the format is independent of the MINDMAP.INI.
Example:	`date` → `12.07.1995` (with MMDEU.DLL installed) `date` → `07/12/95` (with MMENG.DLL installed)
See Also:	datestr, strdate, day, year, month, time, sweekday, smonth
Category:	Intrinsic function.

datestr

Syntax:	datestr(<date>)
Description:	Converts <date> into a string. In most cases this conversion is not necessary, since mindmap always attempts to perform the necessary conversions, if there are inconsistencies in the data types. Note that mindmap uses a language specific notation for the resulting string, depending on the currently selected language DLL.
Parameter(s):	The <date> parameter contains a valid date.
Return Value(s):	A string containing the date, as a string in the format defined in the

	MINDMAP.INI file or with the mask attribute of the component.
Example:	`datestr(edt1)` → `"12.07.1995"` ; where edt1 contains 12.07.1995
	Please note again that, if you forget the datestr function in this example, mindmap would convert date to string automatically, if a string is expected instead of a date.
See Also:	date, strdate, day, year, month, smonth, time, weekday sweekday
Category:	Intrinsic function.

day

Syntax:	day(<date>)
Description:	Returns the numeric value of the day of the month of the calendar date supplied.
Parameter(s):	A string containing the date.
Return Value(s):	A string containing a two-digit number.
Example:	`day(edt1)` → `12` ; where edt1 contains the current date 12.07.1995 (if MMDEU.DLL is installed)
	`day(edt2)` → `12` ; where edt2 contains the current date 07/ 12/ 1995 (if MMENG.DLL is installed)
See Also:	month, smonth, year, date, strdate, datestr, time, weekday, sweekday
Category:	Intrinsic function

DeleteFile

Syntax:	DeleteFile(<text>)
Description:	The function deletes the file, whose file name is contained in <text>.
Parameter(s):	The <text> parameter contains a valid file name or path and file name if the file is not in the current directory.
Return Value(s):	If the function was not successful, one of the following error values will be returned: ▸ File not found ▸ Path not found ▸ Access denied
Example:	This function is especially used to translate language dependent special characters like German umlaut characters to the windows character set. `DeleteFile(edt1)` ; where edt1 contains the text string “FILE1.TXT” or “C:\\FILE1.TXT”. Please note that this function does not support long file names in or Windows NT. `DeleteFile(edt1)` ; where edt1 contains the text string “FILE1.TXT” or “C:\\FILE1.TXT”
See Also:	CopyFile
Category:	Declared function.

exp

Syntax:	exp(<num>)
Description:	Returns the exponential value of the given parameter.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	Exponential value
Example:	`exp (3)` → `20.0855369231877`
See Also:	log, ln
Category:	Intrinsic function.

Format

Syntax:	Format (<text>, <num>)
Description:	This function converts a number into a string. You may select from a variety of format specifiers to control the output.
Parameter(s):	The first parameter specifies the format. It has the following form:

%[-][#][0][width][.precision]type

Each field of the format specification is a single character or number signifying a particular format option. The simplest format specification contains only the percent sign and a type character (for example, %i). The optional fields (in brackets) control other aspects of the formatting. Following are the optional and required fields and their meanings:

Field	Meaning
-	Pad the output value with blanks or zeros to the right to fill the field width, aligning the output value to the left. If this field is omitted, the output value is padded to the left, aligning it to the right.
#	Prefix hexadecimal values with 0x (lowercase) or 0X (uppercase).
0	Pad the output value with zeros to fill the field width. If this field is omitted, the output value is padded with blank spaces.
width	Convert the specified minimum number of characters. The width field is a nonnegative integer. The width specification never causes a value to be truncated; if the number of characters in the output value is greater than the specified width, or if the width field is not present, all characters of the value are printed, subject to the value of the precision field.
precision	Convert the specified

	minimum number of digits. If there are fewer digits in the argument than the specified value, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds the specified precision. If the specified precision is zero or omitted entirely, or if the period (.) appears without a number following it, the precision is set to 1.
type	This field may be any of the following character sequences:

Sequence	Meaning
d, i	Insert a signed decimal integer argument (16-bit).
ld, li	Insert a long signed decimal integer argument (32-bit).
u	Insert an unsigned integer argument (16-bit).
lu	Insert a long unsigned integer argument (32-bit).
x, X	Insert an unsigned hexadecimal integer argument in lowercase or uppercase (16-bit).
lx, lX	Insert a long unsigned hexadecimal integer argument in lowercase or uppercase (32-bit).

	The second parameter is the number to be converted.
Return Value(s):	The function returns the converted string.
Example:	If edt1 contains the value 123, the function will convert the following strings

```
Format("%10I",edt1) → "…….123"
Format("%05I",edt1) → "00123"
Format("%-10I",edt1) → "123……."
Format("%#x",edt1) → "0x7b"
Format("%04x",edt1) → "007b"
```

where . represents a space character.

See Also:	FormatString
Category:	Imported function.

FormatString

Syntax:	FormatString(<text>, <text>)
Description:	This function converts a string. You may select from a variety of format specifiers to control the output.
Parameter(s):	The first parameter specifies the format. It has the following form:

%[-][width][.precision]s

Each field of the format specification is a single character or number signifying a particular format option. The simplest format specification contains only the percent sign and the character s (for example, %s). The optional fields (in brackets) control other aspects of the formatting. Following are the optional and required fields and their meanings:

Field	Meaning
.	Pad the output value with blanks to the right to fill the field width, aligning the output value to the left. If this field is omitted, the output value is padded to the left, aligning it to the right.
Width	Convert the specified minimum number of characters. The width field is a nonnegative integer. The width specification never causes a value to be truncated; if the number of characters in the output value is greater than the specified width, or if the width field is not present, all characters of the value are printed, subject to the value of the precision field.
Precision	Convert the specified maximum number of characters.

The second parameter is the string to be converted.

Return Value(s): The function returns the converted string.

Example:	If edt1 contains the string “Hello”, the function `Format('%10s',edt1)` will return the string “Hello.....” ; where . represents a space character.
See Also:	Format
Category:	Imported function.

frac

Syntax:	frac(<num>)
Description:	This function returns the fractional part of a floating-point value.
Parameter(s):	<num> is a floating-point value
Return Value(s):	A floating-point value

Example:	`frac (3.56)`
	; equals 0.56
	`frac (-5.33)`
	; equals -0.33
	`frac (val(edt1))`
	; equals 0.44 if edt1 contains e. g. “9.44”

See Also:	int, round
Category:	Intrinsic function.

GetFileCount

Syntax:	GetFileCount(<text>)
Description:	Counts the number of files specified

	by <text>. Wildcards are permitted.
Parameter(s):	The parameter <text> contains a valid MS-DOS directory name.
Return Value(s):	An integer is returned.
Example:	`GetFileCount(edt1) →231` ; where edt1 contains the string C:\WINDOWS\*.*.
	When specifying paths always use two backslashes instead of single back-slashes.
See Also:	CopyFile, DeleteFile
Category:	Declared function.

GetHomeDir

Syntax:	GetHomeDir(<text>)
Description:	This function converts the file name defined by <text> into a full path name representing a file in the application's home directory, provided that <text> is not a full path name itself. The application's home directory is the directory where the application EXE file is.
Parameter(s):	<text> Identifies the name of a file and should not contain path and drive specifications.
Return Value(s):	<text> The return value is a string representing a full path name based on the name given through the function's parameter.
Example:	`ReadProfile (“MyApp”,”UserName”,””,GetHomeDir(“DEMO.INI”))`
	the above function reads the user's

	name from the INI file in the same directory where the EXE file is (instead of searching the Windows directory).
See Also:	ReadProfile, WriteProfile
Category:	Declared function.

GetModuleHandle

Syntax:	GetModuleHandle(<text>)
Description:	This function retrieves the handle of the specified module.
Parameter(s):	<text> address of name of module
Return Value(s):	Handle of the module, if the function is successful. Otherwise, it is NULL.
Example:	This function maybe used to verify the existence of a running instance of a program. It will return a non-zero value if the specified module name is already running. For instance GetModuleHandle("Excel") will return a positive integer if Excel has already been launched. Please note that the name of the module is not always identical to the name of the associated EXE file; however, a number of tools are available in the public domain to show a list of all running modules.
Category:	Declared MS-Windows kernel function.

GetParent

Syntax:	GetParent(<num>)
Description:	Retrieves the handle <num> of the given window's parent window (if any). The following hierarchy applies to a mindmap Application running as an executable file: mindmap Application Window Object Area Window while the following hierarchy applies to a mindmap application in the Development Environment: mindmap Frame Window MDIClient class MDI Child Window Object Area Window
Parameter(s):	<num> identifies the window whose parent window handle is to be retrieved.
Return Value(s):	<num> is the handle of the parent window, if the function is successful. Otherwise, it is NULL, indicating an error or no parent window.
Example:	`GetParent(MMWindow))`
See Also:	SetWindowText
Category:	Declared MS-Windows kernel function.

GetTickCount

Syntax:	GetTickCount
Description:	This function returns the number of milliseconds that have elapsed since Windows was started.
Parameter(s):	None.
Return Value(s):	The function returns a numeric value (32-bit) which represents the number of milliseconds since Windows was started.
Example:	`GetTickCount → 22133234`
Category:	Intrinsic function.

GetWindowText

Syntax:	GetWindowText(<num1>, <text>, <num2>)
Description:	This function copies text of the given window's title bar (if it has one) into a buffer. If the given window is a control, the text within the control is copied.
Parameter(s):	<num1> handle of window
	<text> address of buffer for text
	<num2> An integer; maximum number of bytes to copy
Return Value(s):	The length, in bytes, of the copied string, not including the terminating null character. It is zero if the window has no title bar, the title bar is empty, or the <num1> parameter is invalid.
	`GetWindowText(GetParent(MMWin`

	`dow),edt1,256)`
	Since mindmap guarantees the size of a call-by-name argument to be 4096, the third parameter can be any suitable value less than 4096. Under normal circumstances, 256 is sufficient.
See Also:	SetWindowText
Category:	Declared MS-Windows kernel function.

gsum

Syntax:	gsum(<component name>)
Description:	The function computes the sum of all components (that have a value associated with them) placed on top of <component name>. It performs a graphical summation. Instead of expecting values as arguments, this function works on components dragged on top of another component.
Parameter(s):	A component name.
Return Value(s):	A numeric value
Example:	`gsum(rc1)` → `100.2` ; where two components are placed on top of rc1, each having a value of 50.1.
Category:	Intrinsic function.

height

Syntax:	height(<component name>)

Description:	It measures in pixels the height of the specified component.
Parameter(s):	The <component name> parameter is a valid component name.
Return Value(s):	An integer
Example:	`height(rc1)` → `30` ; where the rectangle rc1 is 30 pixels high
See Also:	width
Category:	Intrinsic function

Hex

Syntax:	Hex (<num>)
Description:	This function converts a numerical value into hexadecimal notation.
Parameter(s):	numerical value (only the integer part is used).
Return Value(s):	string containing the converted hexa-decimal value.
Example:	`hex(1024)` → `400` ; returns the string “400”
Category:	MMPSTOOL

hwnd

Syntax:	hwnd(<component name>)
Description:	This function returns the window handle of the window to which the specified component belongs. The following hierarchy applies to a

	mindmap Application running as an executable file:
	mindmap Application Window Object Area Window
	while the following hierarchy applies to a mindmap application in the Development Environment:

	mindmap Frame Window
	    MDIClient class
	        MDI Child Window
	            Object Area Window

Parameter(s):	The <component name> parameter is a valid component name.
Return Value(s):	none
Example:	This function is generally useful in (imported) functions which require the window handle of the window the component is associated with.
See Also:	GetParent, hwndchild
Category:	Intrinsic function.

int

Syntax:	int(<num>)
Description:	This Function returns a floating-point value representing the largest integer that is less than or equal to <num>.
Parameter(s):	The <num> parameter contains any number.
Return Value(s):	Floating-point result; no error return

Example:	`int(edt1) → 10`
	; where edt1 contains 10.5
	`int(edt2) → -10`

; where edt2 contains -9.6.

```
int (2.8)
```
; is 2.000000

```
int (-2.8)
```
; is -3.000000

See Also:	frac, round
Category:	Intrinsic function.

JulianDate

Syntax:	JulianDate (<num>,<num>,<num>)
Description:	This function computes the Julian Date from three numerical values of year, month and date. This function is useful to perform date calculations (e.g. number of days between dates, calculation of the day of week).
Parameter(s):	The first parameter is the year (if this value is less than 100 it is assumed to be the year of 1900+y). The second parameter is the month. The third parameter is the day.
Return Value(s):	returns the Julian Date.
Example:	`JulianDate (44,10,12)` → `2431376`
See Also:	YMDFromJulian
Category:	MMLIB

len

Syntax:	len(<text>)
Description:	Obtains the length of the supplied string.

Parameter(s):	The <text> parameter is a string.
Return Value(s):	Returns an integer.
Example:	`len(edt1) → 7` ; where edt1 contains the string “abcdefg”. If edt1 contains a numeric value instead of a string, the correct expression is the following: `len(str(edt1))`
See Also:	str, substr, strpos, strrepl, upper, lower
Category:	Intrinsic function

ln

Syntax:	ln(<num>)
Description:	Determines the natural logarithm of <num>.
Parameter(s):	The parameter <num> is a floating point value and must be greater than zero.
Return Value(s):	A floating point value as the result or 0 as an error value. Please note that the result of ln(1) is also 0! Also note, that if an error occurs the error log reports this function as log10.
Example:	`ln(1) → 0` `ln(100) → 4.605170` `ln(0.8) → -0.223144` `ln(edt1)` ; edt1 contains a numeri-

	cal value
	ln(val(edt1)) ; edt1 contains a string of numbers
See Also:	log
Category:	Intrinsic function.

log

Syntax:	log(<num>)
Description:	Returns the common logarithm (base 10) of a number.
Parameter(s):	The <num> parameter may have any positive floating point value.
Return Value(s):	A floating point value as the result or 0 as an error value. Please note that the result of log(1) is also 0! Also note, that if an error occurs the error log reports this function as log10.

Example:	`log(11) → 1.041392`
	`log(100) → 2`
	`log(0.8) → -0.096910`
	`log(edt1)` ; edt1 contains a numerical value
	`log(val(edt1))` ; edt1 contains a string of numbers

See Also:	ln
Category:	Intrinsic function.

lower

Syntax:	lower(<text>)
Description:	This function converts all letters in <text> to lowercase.
Parameter(s):	The <text> parameter contains characters.
Return Value(s):	A string containing only lowercase letters.
Example:	`lower(edt1)` → `"abcde"` ; where edt1 contains "ABcDE"
See Also:	upper, str, val
Category:	Intrinsic function.

lstrspn

Syntax:	lstrspn (<text1>,<text2>)
Description:	This function returns the index of the first character in string1 not belonging to string2. This function is useful to remove trailing spaces (or other characters) from strings. Please note that the returned index is zero-based since this is an imported kernel function.
Parameter(s):	Parameter: string to be searched.
	Parameter: characters to be ignored.
Return Value(s):	returns a zero-based index into string1.
Example:	The statement
	`substr(edt1,`

	`lstrspn(edt1,' ')+1)`
	will eliminate all trailing spaces from the string in the input field edt1.
Category:	MMLIB

MakeDir

Syntax:	MakeDir (<text>)
Description:	This function creates the directory specified by the string given in the parameter.
Parameter(s):	Name of the directory to be created. This name should not contain a terminating backslash character. The name may contain a drive specifier.
Return Value(s):	This function returns one of the following numerical values:

0	directory already exists
1	directory has been created
2	cannot create directory (disk may be write protected or specified directory is invalid).

Example:	MakeDir("c:\\example")

MMWindow

Syntax:	MMWindow
Description:	Retrieves the windows handle of the currently active mindmap application window.
Parameter(s):	none
Return Value(s):	An integer value representing the window handle.

Example:	The following operation sets the caption text of the currently active mindmap window.
	`SetWindowText(GetParent(MMWindow),"")`
	Note that, since mindmap application windows are always child windows, their parent is either an MDI child or a tiled window.
See Also:	GetParent, SetWindowText
Category:	Declared mindmap function.

month

Syntax:	month(<date>)
Description:	Extracts the numeric value of the month from the date supplied.
Parameter(s):	A string containing the date.
Return Value(s):	A string containing a two-digit number
Example:	`month(date)`
	`month("12.07.1995")` → `7` (with MMDEU.DLL installed)
	`month("07/12/1995")` → `7` (with MMENG.DLL installed)
	It is important to choose the date format that corresponds to the mindmap language selection.
See Also:	day, week, year, date, strdate, datestr, smonth, sweekday
Category:	Intrinsic function.

ObjectCount

Syntax:	ObjectCount
Description:	It counts the number of components on the currently visible mindmap page.
Parameter(s):	none
Return Value(s):	An integer value.
Example:	`ObjectCount → 12` ; where the page contains 12 components.
See Also:	ObjectValue
Category:	Intrinsic function.

PageCount

Syntax:	PageCount
Description:	It returns the total number of pages in the currently active MM file.
Parameter(s):	none.
Return Value(s):	An integer value.
Example:	`PageCount → 72` ; where the total number of pages in the application is 72.
See Also:	PageNum, ReportPage
Category:	Intrinsic function

PageNum

Syntax:	PageNum
Description:	It returns the page number of the current page of either a mindmap application, a printer template or an output page object. The function returns a value of 1 for the first page.
Parameter(s):	none.
Return Value(s):	An integer value.
Example:	`PageNum` → `6` ; where the function is placed on the 6th page of the application
See Also:	PageCount, ReportPage
Category:	Intrinsic function.

pi

Syntax:	pi
Description:	It computes the value of pi, which is the ratio of the circumference to the diameter of a circle.
Parameter(s):	none.
Return Value(s):	3.14159
Example:	`pi*2` → `6.282`
Category:	Intrinsic function.

PointInObject

Syntax:	PointInObject(<component name>, <x-coordinate>, <y-coordinate>)
Description:	This function determines if a point defined by its x- and y-coordinate lies within a given object.
Parameter(s):	The <component name> parameter is a valid component name. The <x-coordinate> is a horizontal coordinate. The <y-coordinate> is a vertical coordinate.
Return Value(s):	This function returns 1 if the point lies within the given component, otherwise 0.
Example:	Use this function together with the functions GetMouseX and GetMouseY to determine if (and where) the mouse is positioned with respect to a given component.
See Also:	GetMouseX, GetMouseY
Category:	Intrinsic function.

rand

Syntax:	rand
Description:	This function returns a pseudo random value between 0 and 32767.
Parameter(s):	None
Return Value(s):	The value returned is a pseudo random number.

Example:	Use the following conversions if applicable
	`r = rand/32768` → 0 <= r < 1
	`r = 10*rand/32768` → 0 <= r < 10
Category:	Intrinsic function.

ReadProfile

Syntax: ReadProfile(<text1>, <text2>, <text3>, <text4>)

Description: This function returns a string associated with a given entry in an INI file. The INI file is assumed to be in the windows directory if a full path name is not supplied.

Parameter(s):

text1	section name
text2	key name
text3	default value to be returned, if the key does not exist.
text4	name of the INI file.

Return Value(s): A string representing the entry in the INI file or the given default value if the section and keyword could not be found.

Example:

```
ReadProfile
("windows","device","No
printer installed","WIN.INI")
```

returns the currently installed default printer or the message No printer installed if such an entry could not be found.

If the file name is specified without path, it is assumed to be in the Windows directory. Use the GetHomeFile function to specify an .INI file in the

	mindmap home directory.
See Also:	WriteProfile
Category:	Declared function

round

Syntax:	round(<num1>, <num2>)
Description:	Rounds the floating point value <num1>. <num2> is the number of digits to the right of the decimal point.
Parameter(s):	<num1> A floating point value <num2> An integer, the number of decimal digits; optional
Return Value(s):	A floating point value.

Example:	`round(PI,3)` → `3.142`
	`round(PI)` → `3`
	; PI equals 3.14159
	`“$ “+str(round(edt1,2))` → `“$ 132.45”`
	; the contents of edt1 is 132.448123

See Also:	frac, int
Category:	Intrinsic function

SetWindowText

Syntax:	SetWindowText(<num>, <text>)
Description:	This function sets the title (caption bar) of the currently active window to be <text>
Parameter(s):	<num> is the window handle

	<text> is a string
Return Value(s):	This Function does not return a value.
Example:	The function
	`SetWindowText(GetParent(MMWindow),"Print")`
	changes the text in the caption bar of the mindmap Application Window to contain the word Print.
See Also:	GetWindowText, GetParent
Category:	Declared MS-Windows function.

ShowWindow

Syntax:	ShowWindow(<num1>, <num2>)
Description:	The ShowWindow function sets the given window's visibility state.
Parameter(s):	<num1> is the window's handle
	<num2> is an integer; the window visibility flag, defined as follows

0	hide the window
1	show the window
2	minimize the window (show as icon)
3	maximize the window
9	restore the window to its original size and position

Return Value(s):	Returns equal zero, if the window was previously hidden;
	returns nonzero, unequal 0; if the window was previously visible.

Example:	The operation: `ShowWindow(GetParent(MMWindow),2)` minimizes the currently active mind-map application.
See Also:	GetParent, MMWindow
Category:	Declared MS-Windows function.

sign

Syntax:	sign(<num>)
Description:	Determines if <num> is positive or negative.
Parameter(s):	<num> A floating point value
Return Value(s):	A The function returns -1 if the given parameter is less than zero, 1 if the parameter is greater than zero and 0 if the given value is 0.floating point value.

Example:	`sign(-1.23) → -1`
	`sign(1.23) → 1`
	`sign(0) → 0`

See Also:	abs
Category:	Intrinsic function

sin

Syntax:	sin(<num>)
Description:	Returns the sine of an angle that is measured in radians. The formula for converting degrees to radians is radi-

ans=degrees*(/ 180).

Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	A numeric value.
Example:	`sin(edt1)` → `.49999999` ; where edt1 contains 30*(pi/ 180)
See Also:	tan, cos, arcsin, arctan, arccos
Category:	Intrinsic function.

sinh

Syntax:	sinh(<num>)
Description:	Returns the hyperbolic sine of an angle.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	A numeric value.
See Also:	cosh, tanh, arcsinh, arccosh, arctanh
Category:	Intrinsic function.

smonth

Syntax:	smonth(<date>)
Description:	Returns the name of the month of <date> using the installed language library.
Parameter(s):	A date
Return Value(s):	A string

Example:	`smonth(date)` → `"August"`
	; date "08/ 09/ 95"
	(English Format)
	`smonth(edt1)`
	; edt1 contains a date.

See Also:	day, ,week, month, year, date, strdate, datestr, sweekday
Category:	Intrinsic function

sqrt

Syntax:	sqrt(<num>)
Description:	Returns the square root of <num>.
Parameter(s):	The <num> parameter is any expression that yields a non-negative number.
Return Value(s):	A floating point number. Note that this function returns 0 for a square root of a negative number. However, an entry in the parser error window is generated.
Example:	`sqrt(edt1)` → `12` ; where edt1 contains 144.
Category:	Intrinsic function.

str

Syntax:	str(<num>)
Description:	Translates a number <num> into a string.
Parameter(s):	The <num> parameter contains a number value.
Return	A string representation of <num>.

Value(s):	
Example:	`str(edt1) →  "123"` ; where edt1 contains the integer 123.
	`Str(123) -→ "123"`
	`str(-123.45) →"-123.45"`
See Also:	val, strdate, datestr, strpos, substr, strrepl
Category:	Intrinsic function

strdate

Syntax:	strdate(<text>)
Description:	Converts <text> into a valid date if possible. You may use this function to force a conversion where other functions require an argument of the type date.
	Keep in mind that mindmap will attempt to automatically convert strings to date values where dates are required, even if a string is supplied.
Parameter(s):	The parameter <text> contains date.
Return Value(s):	A valid date.
Example:	`strdate(edt1) → 12.07/12/.19975` ; where edt1 contains "12.07/ 12/ .19975".
See Also:	date, datestr, str, substr, strpos, strrepl
Category:	Intrinsic function.

strpos

Syntax:	strpos(<text1>, <text2>)
Description:	Searches for the first occurrence of <text2> in <text1>.
Parameter(s):	<text1> and <text2> contain strings.
Return Value(s):	An integer value with the beginning position (1-based). The value 0 means that nothing was found.

Example:	`strpos(edt1, edt2) → 2` ; where edt1 contains “abcdef” and edt2 contains “bc”.
	 `Strpos(edt1,”file”)`
	 `strpos(“c:\\files\\*.*”,edt2)`

See Also:	str, substr, upper, lower, strdate, datestr, strrepl
Category:	Intrinsic function.

strrepl

Syntax:	strrepl(<text1>, <text2>, <text3>)
Description:	Replaces all occurrences of <text2> with <text3> in <text1>.
Parameter(s):	All parameters contain strings
Return Value(s):	An integer; the number of replacements.
Example:	`strrepl(edt1, edt2, edt3)` ; where edt1 contains “John Miller”, edt2 contains “John”, and edt3 contains “Pete”. After the

replacement, edt1 contains “Pete Miller” and the return value is 1 (for 1 replacement).

See Also:	str, strpos, substr, upper, lower, strdate, datestr
Category:	Intrinsic function.

substr or substring

Syntax:	substr(<text>, <num1>, [<num2>])
	substring(<text>, <num1>, [<num2>])

Description:	Extracts from <text> a substring starting at position <num1> and with a length of <num2> (1-based).
Parameter(s):	<text> contains a string, <num1> designates the starting position, <num2> the desired length of the substring to be extracted. The parameter <num2> is optional; without this value the returned string is from the position to the end of <text>.
Return Value(s):	A string
Example:	`substr(edt1,2,2)` → “bc” `substr(edt1,2)` → “bcdef” ; where edt1 contains “abcdef”. `substr(edt2,1,2) + " " + date` → “Fr 08/04/95” ; where edt2 contains “Friday” and date is “08/ 04/ 95” `substr(sweekday(date),1,2)` →

	`"Fr"` ; where date is "08/ 04/ 95"
See Also:	str, strpos, upper, lower, strdate, dat-estr, strrepl
Category:	Intrinsic function.

sweekday

Syntax:	sweekday(<date>)
Description:	Returns the name of the weekday of <date> using the installed language library.
Parameter(s):	A date
Return Value(s):	A string
Example:	`sweekday(date)` $\rightarrow$ `"Wednesday"` ; date "08/ 09/ 95" (English Format) `sweekday(edt1)` ; edt1 contains a date
See Also:	day, ,week, month, year, date, strdate, datestr, sweekday
Category:	Intrinsic function

tan

Syntax:	tan(<num>)
Description:	Returns the tangent of an angle measured in radians. The formula for converting degrees to radians is radi-ans=degrees * (/ 180).
Parameter(s):	The <num> parameter is any expres-

	sion that yields a number.
Return Value(s):	A floating point.
Example:	`tan(edt1)` → `0.57735` ; where edt1 contains 30*(pi/ 180)
See Also:	sin, cos, arctan, arcsin, arccos
Category:	Intrinsic function.

tanh

Syntax:	tanh(<num>)
Description:	Returns the hyperbolic tangent of an angle.
Parameter(s):	The <num> parameter is any expression that yields a number.
Return Value(s):	A numeric value.
See Also:	sinh, cosh, arcsinh, arccosh, arctanh
Category:	Intrinsic function.

time

Syntax:	time
Description:	Supplies the current system time in 24h notation.
Parameter(s):	none
Return Value(s):	A time.
Example:	time → 12:43:12
See Also:	date

Category:	Intrinsic function.

trim

Syntax:	trim (<text>)
Description:	This function removes trailing spaces from the given string.
Parameter(s):	String to be transformed.
Return Value(s):	This function returns the transformed string.
Example:	`trim('abcdef     ') → 'abcdef'.`
Category:	MMLIB

upper

Syntax:	upper(<text>)
Description:	Returns a string consisting of the uppercase equivalent of the <text> parameter. CharCharacters that lack an uppercase equivalent in the ANSI character set are returned unchanged.
Parameter(s):	The <text> parameter contains a string.
Return Value(s):	A string.
Example:	`upper(edt1) → "ABC/+:?="` ; where edt1 contains "abc/ +:?=".
See Also:	lower, val, str, strpos, substr, strrepl
Category:	Intrinsic

val

Syntax:	val(<text>)
Description:	Returns the numerical equivalent of the supplied <text>, for use with formulas involving numbers or numeric functions.
Parameter(s):	The <text> parameter contains a string representing a number
Return Value(s):	A numerical value.
Example:	`val(edt1)` → `123` ;where edt1 contains the string “123”. `Val(substr(edt2,49,.2))` → `45` ; where edt2 contains the string “1234567890”
See Also:	str, int, frac
Category:	Intrinsic function.

weekday

Syntax:	weekday (<date>)
Description:	This function determines the day-of-week from a given date.
Parameter(s):	The function expects a date value. Use the strdate function to convert a string into a date value.
Return Value(s):	1 = Monday 2 = Tuesday 3 = Wednesday 4 = Thursday 5 = Friday

	6 = Saturday 7 = Sunday
Example:	The function weekday(date) will return the weekday index for to-day. The function `weekday(strdate("12/25/97"))` will return the weekday index for Christmas 1997.
See Also:	sweekday
Category:	MMLIB

width

Syntax:	width(<component name>)
Description:	It measures in pixels, the width of the specified component.
Parameter(s):	The <component name> parameter is a valid component name.
Return Value(s):	An integer.
Example:	`width(rc1)` → `50` ; where the rectangle rc1 is 50 pixels wide.
See Also:	height
Category:	Intrinsic function.

WinHelp

Syntax:	WinHelp (<num>,<text>,<num>,<text>)

Description:	This function directly invokes the windows help system.
Parameter(s):	The first parameter is the window handle of the parent window calling the help system. Please always use the result of the function MMWindow. The second parameter is the name of the help file. The third parameter is one of the following constants: 2 = Close the Windows help system for the specified help file. 3 = Display the contents page of the specified help file. 261 = Show a help screen for the keyword specified in parameter 4. Please note that this keyword must be defined in the help file for this function to be successful. The fourth parameter is the keyword if the third parameter is 261. Otherwise this parameter should be an empty string.
Return Value(s):	The return value is non-zero if the function was successful. Otherwise zero.
Example:	`WinHelp(mmwindow,'EXAMPLE.HLP',261,'Options')`
Category:	MMPARSE.MNC

WriteProfile

Syntax:	WriteProfile(<text1>, <text2>, <text3>, <text4>)
Description:	This function places a string associated with a given entry into an INI file. The INI file is assumed to be in

	the windows directory if a full path name is not supplied.	
Parameter(s):	text1	section name
	text2	key name
	text3	desired value for the given key
	text4	name of the INI file.
Return Value(s):	none	
Example:	The function:	

```
WriteProfile
("MyApp","UserName",edt1
,"MYAPP.INI")
```

writes the contents of the input field with the name edt1 to the file MYAPP.INI in the windows directory. Assuming that edt1 contains the name of the user, his name may later be retrieved by the operation:

```
ReadProfile
("MyApp","UserName","","MYAPP
.INI")
```

which is normally used in a value assignment to the input field edt1, again.

See Also:	ReadProfile
Category:	Declared function

xpos

Syntax:	xpos(<component name>)
Description:	It returns the upper left position of the specified <component name> measured in pixels from the top of the screen.

Parameter(s):	The <component> contains a valid component name.
Return Value(s):	An integer.
Example:	`xpos(edt1)` → `56` ; where edt1 is positioned at the 56th pixel from the top of the screen.
See Also:	ypos
Category:	Intrinsic function.

year

Syntax:	year(<date>)
Description:	Returns the year of the calendar date supplied.
Parameter(s):	A string containing the date.
Return Value(s):	A string containing a four-digit num-ber.
Example:	`year(date)` → `"1995"` `year(edt1)` → `"1995"` ; where edt1 contains the current date 12.07.1995 or 12.07.95
See Also:	day, month, date, strdate, datestr, sweekday, smonth
Category:	Intrinsic function

YMDFromJulian

Syntax:	YMDFromJulian (<num>)
Description:	This function converts a Julian Date into a mindmap date value. This func-

	tion is useful to reconvert the result of date calculations (e.g. number of days between dates, calculation of the day of week).
Parameter(s):	The parameter is a numeric value representing a Julian date.
Return Value(s):	The return value is a mindmap date. Use the datestr function to convert this date into a string.
Example:	`datestr(YMDFromJulian(JulianDate(44,10,12)+14))` → `24.10.1944`
See Also:	JulianDate, datestr
Category:	MMLIB

ypos

Syntax:	ypos(<component name>)
Description:	It returns the upper left position of the specified <component name> measured in pixels from the left of the screen..
Parameter(s):	The <component name> contains a valid component name.
Return Value(s):	An integer.
Example:	`ypos(edt1)` → `152` ; where edt1 is positioned at the 152nd pixel from the left of the screen.
See Also:	xpos
Category:	Intrinsic function

Component Specific Functions

Some components register their own functions. These are always available, as long as the corresponding component library (*.MDL) has been loaded by mindmap.

Database Functions

The database functions are declared by MMBASE.MDL, that is part of the mindmap standard installation. If this file is loaded as specified in the MINDMAP.INI, the following functions are available:

dbBaseName

Syntax:	dbBaseName(<database>)
Description:	This function determines the name of the database, or in case of ODBC, the name of the data source to which <database> belongs.
Parameter(s):	The <database> parameter corresponds to a database name, as defined in mindmap.
Return Value(s):	A string.
Example:	`dbBaseName(db1)`
See Also:	dbTableName, dbFieldName
Category:	Declared by MMBASE.MDL

dbCurrentRow

Syntax:	dbCurrentRow(<database>)
Description:	This function calculates the current record number in the current result set.. The first record number is 1.
Parameter(s):	The <database> parameter corresponds to a database name, as defined in mindmap.
Return Value(s):	An integer.
Example:	`dbCurrentRow(db1)`
	`dbCurrentRow(db1)+"/" +dbRowCount(db1)` ;this expression displays the current position in a result set, e. g. 11/ 131 would mean, the 11th record of 131 records
See Also:	dbRowCount
Category:	Declared by MMBASE.MDL

dbFieldCount

Syntax:	dbFieldCount(<database>)
Description:	Determines the number of columns of the database table defined by the database object supplied as parameter.
Parameter(s):	The <database> parameter corresponds to a database name, as defined in mindmap.
Return Value(s):	An integer.

Example: `dbFieldCount(db1)` → `13`
; if the table db1 consists of 13 columns

Category: Declared by MMBASE.MDL

dbFieldName

Syntax: dbFieldName(<database>,<num>)

Description: This function determines the name of the column in the database table <database> at position <num> (1-based).

Parameter(s): The <database> parameter corresponds to a database name, as defined in mindmap.

Return Value(s): A string.

Example: `dbFieldName(db1,2)` → `"FirstName"`
if the second field in the database object is named FirstName.

See Also: dbBaseName, dbTableName

Category: Declared by MMBASE.MDL

dbGetDate

Syntax: dbGetDate(<database>,<text>)

Description: This function converts a date into the correct format depending on the database used. Use this function in database queries issued through SQL Exec

	commands, e.g. that are not generated automatically through the database command Search for Fields.
Parameter(s):	<database> name of the database component <text> name of the component containing the date that has to be converted
Return Value(s):	A string

Example:	`dbGetDate(db1,edt1) → {(d '1995-08-08'})` ; for an ODBC data source
	
	The following statement may be used in a SQL Select command, assuming that edtBirth has a value of "01/01/1965":
	
	`"where BirthDate > " + dbGetDate(dbEmployees,edtBirth)`
	
	The next statement finds all database records where the column BirthDate lies between two boundaries edtBirthBegin and edtBirthEnd:
	
	`"where BirthDate between " + dbGetDate(dbEmployees,edtBirthBegin) + " and " + dbGetDate(dbEmployees,edtBirthEnd) + ")"`

Category:	Declared by MMBASE.MDL

dbIsOpen

Syntax:	dbIsOpen(<database>)
Description:	This function tests if the database <database> is open, in which case a

	conconnect operation to the database has been
	sucsuccessful and a cursor has been es
	tablished.tablished.
Parameter(s):	The <database> parameter corresponds to a database name as defined in mindmap.
Return Value(s):	0, if database is not open 1, if database is open
Example:	`dbIsOpen(db1) → 0` ; meaning that the database db1 is not currently opened.
Category:	Declared by MMBASE.MDL

dbRowCount

Syntax:	dbRowCount(<database>)
Description:	This function supplies the number of records in the result set. Please not that it depends on the database driver how this function works. In general, most ODBC drivers are not capable of returning the number of rows in the result set, unless the last record in the result set has been fetched. This implies that, to be safe, a “go to last record” command should be executed before using this function. The function returns -1 if the number of records in the database is unknown.
Parameter(s):	The <database> parameter corresponds to a database name, as defined

	in mindmap.
Return Value(s):	An integer.
Example:	`dbRowCount(db1)`
	`dbCurrentRow(db1)+"/"+ dbRowCount(db1)` ;this expression displays the current position in a result set, e. g. 11/ 131 would mean, the 11th record of 131 records
See Also:	dbCurrentRow
Category:	Declared by MMBASE.MDL

dbSQLSearch

Syntax:	dbSQLSearch(<database>)
Description:	Retrieves the "WHERE"-part of a SQL SELECT statement, according to the components to which the database is connected and for which the "search" option has been set. Please note that the "WHERE"-part is also dependent on the "Search Mode"-settings of an input field,. This is set by setting the attribute symbolized by the magnifier glass on the attribute toolbox of the component.
Parameter(s):	The <database> parameter corresponds to a database name, as defined in mindmap.
Return Value(s):	A string.
Example:	`dbSQLSearch(db1) →` `((ADDNO = 2 and COMPANY = "MGM") and (LASTNAME like`

	`“R%” or LASTNAME like “S%”))`
	In this example there are 3 input fields connected to db1:
	edt1 with contents 2 and Search Mode is <f> = <a>
	edt2 with contents “MGM” and Search Mode is <f> = <a>
	edt3 with contents “R;S” and Search Mode is <f> like “<a>%”
Category:	Declared by MMBASE.MDL

dbTableName

Syntax:	dbTableName(<database>)
Description:	This function determines the name of the table in the database which is represented by the component <database> on the screen.
Parameter(s):	The <database> parameter corresponds to a database name, as defined in mindmap.
Return Value(s):	A string.
Example:	`dbTableName(db1)` → `Address`
See Also:	dbBaseName, dbFieldName
Category:	Declared by MMBASE.MDL

Input Field Functions

The data table functions are declared by MMEDIT.MDL and/ or MMEDIT.MDL, that is part of the mindmap standard installation. If these files are loaded as specified in the MINDMAP.INI, the following functions are available:

edtGetCol

Syntax:	edtGetCol(<input field >)
Description:	This function determines the column position of the caret in an input field. You may want to use this function to extract a portion of the contents of an input field corresponding to the currently selected text. Please note that this function is valid only if the input field is visible. It is not applicable for input fields that are not on the active page.
Parameter(s):	The <input field> parameter is the name of an input field.
Return Value(s):	This function returns the 1-based column number which identifies the caret position or 0 if the input field is invalid or does not exist.
Example:	`edtGetCol (edt1) → 4` ; where the cursor has been used to select the 4th column in the input field edt1
See Also:	edtGetRow, edtSetPos
Category:	MMEDIT.MDL

edtGetRow

Syntax:	edtGetRow(<input field >)
Description:	This function determines the row number of the caret in a multiline input field. For a single line input field this function will always return 1. You may want to use this function to extract a portion of the contents of an input field corresponding to the currently selected text. Please note that this function is valid only if the input field is visible. It is not applicable for input fields that are not on the active page.
Parameter(s):	The <input field> parameter is the name of an input field.
Return Value(s):	This function returns the 1-based row number which identifies the caret position or 0 if the input field is invalid or does not exist.
Example:	`edtGetRow (edt1) → 3` ; where the cursor has been put into the 3rd row of the multiline input field edt1
See Also:	edtGetCol, edtSetPos
Category:	MMEDIT.MDL

edtSetPos

Syntax:	edtSetPos(<input field >, <row number>, <column number>)
Description:	This function sets the caret to the po-

	sition specified by the given row and column number. Please note that this function is valid only if the input field is visible. It is not applicable for input fields that are not on the active page.
Parameter(s):	The <input field> parameter is the name of an input field. The <row number> defines the row to which the caret is to be placed. This value must always be 1 for single line input fields. The <column number> defines the column to which the caret is to be placed.
Return Value(s):	This function does not return a value.
Example:	`edtSetPos (edt1, 2, edt2)` → caret set to row 2, column 5 ; where edt1 is a multiline input field and edt2 contains the number 5.
See Also:	edtGetCol, edtGetRow
Category:	MMEDIT.MDL

Data Table Functions

The data table functions are declared by MMDATA.MDL, that is part of the mindmap standard installation. If these files are loaded as specified in the MINDMAP.INI, the following functions are available:

Columns

Syntax:	Columns (<data table name>)
Description:	Determines the number of columns in a data table.
Parameter(s):	Name of a data table
Return Value(s):	Number of columns as an integer value
Example:	`Columns (dt1)` $\rightarrow$ `12` ; where the data table dt1 contains 12 columns
See Also:	Rows
Category:	MMDATA.MDL

CurrentCol

Syntax:	CurrentCol (<Object name>)
Description:	Returns the number (1-based) of the column of a data table which has the input focus.
Parameter(s):	Name of a data table.
Return Value(s):	Number of the selected column as an integer.
Example:	`CurrentCol (dtb1)` $\rightarrow$ `4` ; where the cursor has been used to select the 4th column
Category:	MMDATA.MDL

CurrentRow

Syntax:	CurrentRow(<datatable>)
Description:	Determines the number of the current row (1-based).
Parameter(s):	<datatable> component name
Return Value(s):	An integer
Example:	`CurrentRow(tbl1) → 5` ; if the 5th row of tbl1 is selected `str(CurrentRow(tbl1))+ " / "+ str(Rows(tbl1)) → "5 / 501"` ; displays the current row in the datatable tbl1, e. g. 5/ 501 would mean, the 5th row of 501 rows.
See Also:	Rows
Category:	Declared by MMDATA.MDL

FirstMarkedRow

Syntax:	FirstMarkedRow(<datatable>)
Description:	Retrieves the number of the first marked row in a multiple selection data table (1-based).
Parameter(s):	<datatable> component name
Return Value(s):	An integer for the row position or a negative number if nothing is selected.
Example:	`FirstMarkedRow(tbl1) → 5`

	; the 5th, 7th and 8th row of tbl1 are selected
See Also:	IsRowMarked
Category:	Declared by MMDATA.MDL

IsRowMarked

Syntax:	IsRowMarked(<datatable>,<row>)
Description:	Checks if the row number <row> is highlighted (1-based). Please note that this is only defined for multiple-selection data tables. The function returns a negative value if an error has occurred.
Parameter(s):	<datatable> component name <row> An integer; number of the row
Return Value(s):	0 ; not highlighted 1 ; highlighted
Example:	`IsRowMarked(tbl1,3)` ; checks if the 3rd row of data table tbl1 is highlighted
See Also:	FirstMarkedRow
Category:	Declared by MMDATA.MDL

Rows

Syntax:	Rows(<datatable>)
Description:	Determines the number of rows in the Datatable <datatable>
Parameter(s):	<datatable> component name
Return Value(s):	An integer; number of rows

Example:	`Rows(tbl1) -> 124` ; the Datatable tbl1 consists of 124 rows. `str(CurrentRow(tbl1))+ ' / '+ str(Rows(tbl1)) -> '3 / 124'` ; displays the current row, 3, in the data table tbl1, which contains 124 rows.
See Also:	CurrentRow
Category:	Declared by MMDATA.MDL

SetDataTable

Syntax:	SetDataTable(<datatable>,<row>,<column>,<text>)
Description:	Sets the value <text> into the data table at position [<row>,<column>].
Parameter(s):	<datatable> component name <row> An integer <column> An integer <text> A string
Return Value(s):	none
Example:	`SetDataTable(tbl1,3,2,'Hello world!')` ; The string 'Hello world!' is set into the data table tbl1 at position [3,2] which means the 3rd row, 2nd column
Category:	Declared by MMDATA.MDL

List Box/ Combo Box functions

The list box and combo box functions are declared by MMCOMBO.MDL, that is part of the mindmap standard installation. If this file is loaded as specified by the MINDMAP.INI, the following functions are available:

CursorPos

Syntax:	CursorPos(<list box/ combo box>)
Description:	Determines the number of the current row (1-based).
Parameter(s):	<list box/ combo box> component name
Return Value(s):	An integer
Example:	`CursorPos(lst1) → 3` ; the 3rd row is selected in list box lst1 `str(CursorPos(lst1))+" / " +str(LineCount(lst1)) →` `"5 / 104"` ;this expression displays the navigation in the list lst1, e. g. 5/ 104 would mean that the cursor is positioned at the 5th entry of 104 entries
See Also:	LineCount, SelCount
Category:	Declared by MMCOMBO.MDL

LineCount

Syntax:	LineCount(<list box>)

Description:	Determines the number of all entries in the list box.
Parameter(s):	<list box> component name
Return Value(s):	An integer
Example:	`LineCount(lst1) → 382` ; list box lst1 has 382 entries `str(CursorPos(lst1))+" / " +str(LineCount(lst1)) → "5 / 104"` ;this expression displays the navigation in the list box lst1, e. g. 5/ 104 would mean that the cursor is positioned at the 5th entry of 104 entries
See Also:	CursorPos, SelCount
Category:	Declared by MMCOMBO.MDL

SelCount

Syntax:	SelCount(<list box >)
Description:	Determines the number of selected rows in a list box if the list box has either the multiple selection or the extended selection style.
Parameter(s):	<list box/combo box> component name
Return Value(s):	An integer
Example:	`SelCount(lst1) → 3` ; 3 entries are selected
See Also:	CursorPos, LineCount
Category:	Declared by MMCOMBO.MDL

Output page functions

The report functions are declared by MMREPORT.MDL, which is part of the mindmap standard installation. If this file is loaded as specified in the MINDMAP.INI, the following functions are available:

ReportPage

Syntax:	ReportPage(<component name>)
Description:	Retrieves the currently active page number of a report.
Parameter(s):	<component name> name of the report
Return Value(s):	An integer
Example:	`ReportPage(rpt1)` → `4`
See Also:	ReportPageCount, PageNum
Category:	Declared by MMREPORT.MDL

ReportPageCount

Syntax:	ReportPageCount(<component name>)
Description:	Retrieves the number of pages in a report.
Parameter(s):	<component name> name of the report
Return Value(s):	An integer
Example:	`ReportPageCount(rpt1)` → `14`
See Also:	ReportPage, PageCount

Category:	Declared by MMREPORT.MDL

VBX Functions

These functions are available if (i) a VBX control is installed and (ii) the mindmap VBX library (MMVBX.MDL) has been loaded.

GetParam

Syntax:	GetParam(<VBX component>, <index>)
Description:	This function is valid only during the response to events generated by a VBX component. It retrieves a parameter which has been provided by the VBX component.
Parameter(s):	The <VBX component> parameter references the VBX component which generated the event. The <index> parameter specifies which of the list of parameters has to be retrieved (1-based).
Return Value(s):	The function returns the appropriate parameter as a string regardless of what type the VBX component has assigned to it.
Example:	This function can be used in cases where a VBX event supplies parameters that a Visual Basic program would normally be able to interpret. If the documentation for a VBX control (placed as vbx1 in mindmap) defines an event through

	`Sub Sample_Change ([Index As Integer])`
	the function GetParam(vbx1,1) will return the value of Index as a string.
See Also:	GetProp, SetProp
Category:	MMVBX.MDL

GetProp

Syntax:	GetProp (<VBX component name>,<text>)
Description:	This function returns the value of a property of the given VBX component. The property is defined by the second parameter.
Parameter(s):	The <VBX component> name as it has been declared in the application.
	A <text> parameter specifying the name of the property as the VBX has declared it.
Return Value(s):	A value corresponding to the property value.
Example:	`GetProp (vbx1,"Height")` → `37` ;where the height of the placed VBX instances equals 37 pixels.
See Also:	SetProp, GetParam
Category:	Declared in MMVBX.MDL

SetProp

Syntax:	SetProp (<VBX component name>,<text>,<text>)

Description:	This function directly manipulates the property of a VBX component.
Parameter(s):	The first parameter is the name of the VBX component. The second parameter is the name of the property to be changed. The third parameter is the property value as string. Please convert numeric values to a string even if the specified property requires a numeric value.
Return Value(s):	none.
See Also:	GetProp
Category:	MMVBX.MDL

MCI Functions

These functions are available if (i) an MCI driver is installed for Windows and (ii) the MCI mindmap library (MMVFW.MDL) has been loaded.

mciGetAlias

Syntax:	mciGetAlias (<Multimedia component name>)
Description:	This function returns the currently used alias name for the given multimedia component. This name may be required by certain MCI driver specific functions.
Parameter(s):	Name of a multimedia component.
Return Value(s):	This function returns a string containing the MCI alias name.

Example:	`mciGetAlias(mci1)` → `3061` ; where 3061 represents some (arbitrary) internal identifier, pointing to the component mci1.
See Also:	mciSendString
Category:	MMVFW.MDL

mciGetFileName

Syntax:	mciGetFileName (<Multimedia component name>)
Description:	This function returns the name of the file that has been opened by the given multimedia component.
Parameter(s):	Name of a multimedia component.
Return Value(s):	This function returns a string containing the name of a multimedia file.
Example:	`mciGetFileName(mci1)` → `C:\VFW\MINDMAP.AVI`
Category:	MMVFW.MDL

mciGetLength

Syntax:	mciGetLength (<Multimedia component name>)
Description:	This function returns the length of a multimedia component. The unit of this value depends on the currently selected multimedia device. (A Video for Windows file (*.AVI) returns the length in Frames).
Parameter(s):	Name of a multimedia component.
Return	Numerical value specifying the length

Value(s):	of the file in device specific units.
Example:	The multimedia command “Seek” with a value as of `mciGetLength(mci1) / 2` will play the mci component named mci1 from its middle position.
See Also:	mciGetLength
Category:	MMVFW.MDL

mciGetMediaName

Syntax:	mciGetMediaName (<Multimedia component Object name>)
Description:	This function returns the media name of the currently selected media, if you load a new mci file at run time, i.e. via drag&drop. The value of the return parameter is dependent on the type of media device which is loaded.
Parameter(s):	Name of a multimedia component.
Return Value(s):	Generally a file name, but this depends on the media device type.
Category:	MMVFW.MDL

mciGetMode

Syntax:	mciGetMode (<Multimedia component name>)
Description:	This function returns a string which describes the current state of the multimedia component. Please note that this string depends on the type of MCI driver.
Parameter(s):	Name of a multimedia component.

Return Value(s):	A string describing the state of the component. For a Video for Windows component (*.AVI) this function may return "stopped" "playing"

Example:	`mciGetMode (mci1) →`
	`"stopped"`
	; where the playing of the file has been stopped.

Category:	MMVFW.MDL

mciGetPosition

Syntax:	mciGetPosition (<Multimedia component name>)
Description:	This function returns the current position of a multimedia component. The unit of this value depends on the currently selected multimedia device. (A Video for Windows file (*.AVI) returns the position in Frames).
Parameter(s):	Name of a multimedia component.
Return Value(s):	Numerical value specifying the current position of the file in device specific units.
Example:	`mciGetPosition (mci1) → 124` ; where the AVI file is currently positioned to frame number 124. You may want to use this function to continuously display the position of a playing multimedia device by assigning the result of this function to a text component through a link on the multimedia event "Position Changed".
See Also:	mciGetPositionString

Category:	MMVFW.MDL

mciGetPositionString

Syntax:	mciGetPositionString (<Multimedia component name>)
Description:	This function returns a string describing the current position of a multimedia component. The contents of this string depends on the currently selected multimedia device.
Parameter(s):	Name of a multimedia component.
Return Value(s):	A string describing the current position of the file in device specific format.
Example:	You may want to use this function to continuously display the position of a playing multimedia device by assigning the result of this function to a text component through a link on the multimedia event “Position Changed”.
See Also:	mciGetPosition
Category:	MMVFW.MDL

mciGetRepeat

Syntax:	mciGetRepeat (<Multimedia component name>)
Description:	This function determines if the given mulmultimedia component is in repeat mode. ReRepeat mode means that the multimedia comcomponent is played repeatedly. Repeat mode is set through

	a multimedia link com
	mand.mand.
Parameter(s):	Name of a multimedia component.
Return Value(s):	0 if not in repeat mode, 1 if in repeat mode.
Example:	`mciGetRepeat (mci1)` → `0` (not in repeat mode)
Category:	MMVFW.MDL

mciGetSpeed

Syntax:	mciGetSpeed (<Multimedia component name>)
Description:	This function returns the currently selected output speed of a multimedia component. The value is returned in percent of the normal output speed. Therefore 1000 means normal speed, 500 means half speed.
Parameter(s):	Name of a multimedia component.
Return Value(s):	Numerical value specifying the percentage of normal speed.
Example:	`mciGetSpeed (mci1)` → `1000`
Category:	MMVFW.MDL

mciGetStart

Syntax:	mciGetStart (<Multimedia component name>)
Description:	This function returns the start position of a multimedia component. The unit of this value depends on the currently selected multimedia device. (A

	Video for Windows file (*.AVI) usually returns zero).
Parameter(s):	Name of a multimedia component.
Return Value(s):	Numerical value specifying the start position.
Example:	`mciGetStart (mci1)` → `0` ; where mci1 is an .AVI file.
See Also:	mciGetLength
Category:	MMVFW.MDL

mciGetVolume

Syntax:	mciGetVolume (<Multimedia component name>)
Description:	This function returns the sound volume of a multimedia component, if applicable. A value of 1000 specifies the normal output volume. Specify lower values to decrease the volume and higher values to increase the volume.
Parameter(s):	Name of a multimedia component.
Return Value(s):	Numerical value specifying the volume level.
Example:	`mciGetVolume(mci2)` → `1000` ; where mci2 points to a .WAV file.
Category:	MMVFW.MDL

mciSendString

Syntax:	mciSendString (<text>,<text>,<Numerical

	value>,<Numerical value>)
Description:	This function is imported directly from the Windows Multimedia support DLL. It sends commands immediately to a multimedia device and returns the result string. You may want to use this function to control media devices not supported by the multimedia component.
Parameter(s):	The first string is the command to be sent to the multimedia device. The second parameter should be the name of an input field which will receive the response to the command. The third parameter actually specifies the maximum length of the response. It should be set to 255. The fourth parameter must be 0.
Return Value(s):	This function returns zero if it was successful. Otherwise, a device specific error code is returned. Please refer to the documentation of the multimedia device.
Example:	`mciSendString ("play " +mciGetAlias(mci1), edt4, 255, 0)` will start to play the .AVI file referenced in mci1.
See Also:	mciGetAlias
Category:	MMPARSE.MNC

Registering External Functions

Following is an example of how mindmap can read and write strings to and from INI files.

Please verify that your MMPARSE.MNC file includes the following two lines and that the file MMPSTOOL.DLL is in your mindmap home directory, or at least in a path pointed to by the PATH environment variable.

```
declare WriteProfile lib 'mmpstool.dll' alias
'WriteProfile' (string, string, string, string)
as integer
```

```
declare ReadProfile lib 'mmpstool.dll' alias
'ReadProfile' (string, string, string, string) as
string
```

```
declare GetHomeDir lib 'mmpstool.dll' (string) as
string
```

The parameters are interpreted as follows:

ReadProfile

1st string	section name in the INI file.
2nd string	key name in the section.
3rd string	default return value if the section or the key are missing.
4th string	name of the INI file. If you omit the path in this string, the windows directory is assumed.

WriteProfile

1st string	section name in the INI file.
2nd string	key name in the section.
3rd string	value to write into the INI file.
4th string	name of the INI file. If you omit the path in this string, the windows directory is assumed.

GetHomeDir

1st string	file name without path specification.

Now create a value assignment and select as the target, an input field or a text field to contain the name of the actual entry. Enter the following line into the value editor of the assignment message:

```
ReadProfile
('System','Test','C:\AUTOEXEC.BAT','DEMO.INI')
```

```
WriteProfile
('System','Test','C:\TEST.BAT','DEMO.INI')
```

If you want to place the INI file into the mindmap home directory, you may replace the specifier 'MINDMAP.INI' with GetHomeDir('MINDMAP.INI').

```
ReadProfile
('System','Test','C:\AUTOEXEC.BAT',GetHomeDir('DE
MO.INI'))
```

```
WriteProfile
('System','Test','C:\TEST.BAT',GetHomeDir('DEMO.I
NI'))
```

During load-time, mindmap reads the file MMPARSE.MNC which allows the registration of functions from user-supplied DLLs into the parser. Subsequently, such functions may be used in value assignment messages, conditions, and all other places where strings are parsed.

Please note that there are some restrictions related to the registration of external functions:

1. All external functions must follow the FAR PASCAL calling scheme. (This convention implies that parameters are pushed on the stack from left to right. This also means that variable length parameter lists are not available. The called function is responsible for cleaning up the stack. FAR implies that the function is called via a 32-bit address.)
2. Only four types of parameters are supported: ASCIIZ-String (string), 16-bit integer (integer) , 32-bit integer (long), and 8 byte floating point (double).
3. Integer, long and double are passed as 'call by value' if not otherwise declared. Numeric data types can be forced to follow the 'call by name' scheme by adding the byname modifier.

4. Strings are passed as ‘call by name’ as FAR. If a DLL function passes a string back to the caller, the buffer pointed to by the string is defined to hold al least4096 characters, including the terminating null. If the component (e.g. an input field) already contains a larger string this string is passed in its complete length. Remember that the called function cannot reallocate this pointer.

5. An external function may return either a 16-bit integer, a 32-bit integer or a FAR pointer to a string. In the latter case, the buffer pointed to by the return value must either exist in the default data segment of the called function or be allocated in local or global memory. It must not be allocated on the stack (automatic variable).

The following sample shows how to register the WinHelp function from the Windows Kernel:

```
declare WinHelp lib ‘user’ (integer, string, in-
teger, long) as integer
```

The next example shows how a floating-point parameter can be passed back to mindmap:

```
declare SampFunc lib ‘sample.dll’ (integer, inte-
ger, double byname) as integer
```

The corresponding declaration in the C-source for sample.dll would then look like:

```
short __far __pascal __export SampFunc (short x,
short y, double __far* pReturn);
```

Additionally, the system allows constants to be defined through the following syntax:

```
const SW_HIDE = 0
```

Please note that syntax errors in the MNC file are reported through entries in the MMERROR.LOG file, which may be examined by using the MMINFO utility in the Options/Preferences menu.

Also note that if mindmap detects an error in an MNC file the interpretation is abandoned.